JPL Publication 88-31

Hypercube Matrix Computation Task

Report for 1986-1988

Ruel H. Calalo
William A. Imbriale
Nathan Jacobi
Paulett C. Liewer
Thomas G. Lockhart
Gregory A. Lyzenga
James R. Lyons
Farzin Manshadi
Jean E. Patterson

August 1, 1988

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL Publication 88-31

Hypercube Matrix Computation Task

Report for 1986-1988

Ruel H. Calalo
William A. Imbriale
Nathan Jacobi
Paulett C. Liewer
Thomas G. Lockhart
Gregory A. Lyzenga
James R. Lyons
Farzin Manshadi
Jean E. Patterson

August 1, 1988

NASA
National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

ABSTRACT

A major objective of the Hypercube Matrix Computation effort at the Jet Propulsion Laboratory (JPL) is to investigate the applicability of a parallel computing architecture to the solution of large-scale electromagnetic scattering problems. Three scattering analysis codes are being implemented and assessed on a JPL/California Institute of Technology (Caltech) Mark III Hypercube. The codes, which utilize different underlying algorithms, give a means of evaluating the general applicability of this parallel architecture. The three analysis codes being implemented are a frequency domain method of moments code, a time domain finite difference code, and a frequency domain finite elements code. These analysis capabilities are being integrated into an electromagnetics interactive analysis workstation which can serve as a design tool for the construction of antennas and other radiating or scattering structures.

This document is a summary of the first two years of work on the Hypercube Matrix Computation effort. It includes both new developments and results as well as work previously reported in the "Hypercube Matrix Computation Task: Final Report for 1986-87" (JPL Publication 87-18).

ACKNOWLEDGEMENTS

The work contained in this report has been done in the Jet Propulsion Laboratory's Telecommunications Science and Engineering Division. Two sections of this division have joined together on this work to provide unique expertise. One section is the Tracking Systems and Applications Section, where much of the Mark III Hypercube hardware and software development, as well as parallel algorithm development for scientific applications, takes place. The other is the Radio Frequency and Microwave Subsystems Section, which has provided much experience in antenna design and electromagnetics scattering analysis.

We wish to acknowledge the most helpful consultation received from the University of Illinois, Urbana-Champaign Electrical Engineering and Computer Science Department. In particular, we would like to express thanks to Professor Raj Mittra, Dr. Andrew Peterson, and Stephen Gedney in the Electromagnetics Laboratory. They have provided us with much insight in the formulation of the finite element problem as well as a finite element code. We also wish to thank Professor Alan Taflove at Northwestern University for the very useful discussions and for providing us with a finite difference code.

This work was performed under NASA contract NAS7-918, Task Order RE232, Amendment No. 8.

CONTENTS

Figures

Tables

SECTION I

INTRODUCTION

A major objective of the Hypercube Matrix Computation task is to investigate the applicability of a parallel computing architecture to the implementation and solution of large-scale electromagnetic scattering problems by implementing codes which would encompass different numerical algorithms. Several analysis codes are being assessed on a parallel computing Mark III Hypercube. The first code which has been implemented is the frequency domain method of moments solution, Numerical Electromagnetics Code (NEC-2), developed at Lawrence Livermore National Laboratory. The second code is a time domain finite difference solution to Maxwell's equations. A third code currently being developed is a 2-dimensional finite element technique.

To assess the applicability of the codes we must be able to characterize the performance. Several different measures may be applied when assessing this performance. The first measure compares two things: first, the problem size possible using the hypercube with 128 megabytes of dynamic memory employed in a 32-node configuration, and second, the problem size possible using a more typical sequential user environment. Another measure of performance is the computational speedup attained by the parallel architecture. The speedup can be measured by three methods: 1) comparing the CPU times for key components of the code running on the 32-node Mark III Hypercube with the CPU times for the same code components running, for instance, on a VAX 11/750; 2) comparing the times for the code running in 32 nodes with the times for the code running in a single node; and 3) comparing the times when the problem size per node is fixed and the number of active nodes is varied. This last measure of speedup assesses the scalability of the code to larger hypercube configurations.

Having developed the analysis capabilities, we have as our next objective the integration of the codes into an electromagnetics interactive analysis environment. A workstation has been designed to facilitate all three analysis functions: 1) graphical specification of the structure to be analyzed, 2) hypercube execution of analysis codes, and 3) graphical display of the output.

SECTION II

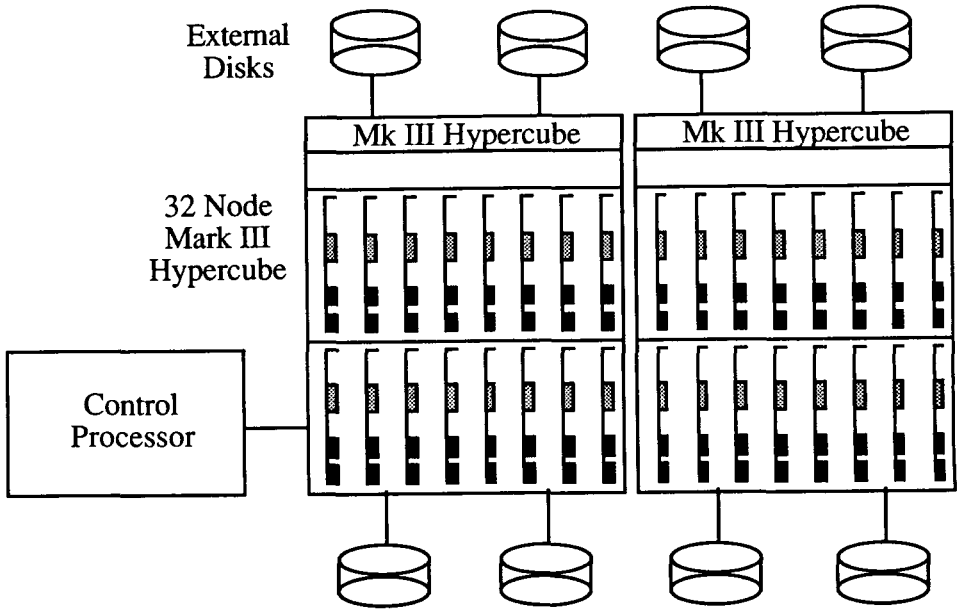
THE MARK III HYPERCUBE

A. INTRODUCTION

Recent advances in high speed microprocessor technology and in methods to combine large numbers of these processors into concurrent structures introduce cost-effective means for solving massive computing problems. A number of different schemes for the connectivity as well as for the components of the computing elements for such concurrent architectures have been suggested. Some machines utilize massive numbers of small-grain (i.e., small memory size) computing elements; others concentrate large-grain computing capability in a modest number of distributed computing elements. One such architecture is the Jet Propulsion Laboratory (JPL)/California Institute of Technology (Caltech) Hypercube. A hypercube is a connectivity scheme which can be viewed as an array of N nodes where each node is capable of communicating directly with $n = \log_2 N$ neighboring nodes along the edges of an n-dimensional cube. The JPL/Caltech Hypercube is now in its third generation of development. At this time, configurations consist of up to 64 nodes with a 128-node Mark III now being connected.

B. HARDWARE

The Mark III Hypercube node has a pair of Motorola 68020 processors--one is the main application processor and the second is the communication processor (Figure 2.1). The communication processor handles internode communications generally without the need to interrupt the main processor. The hypercube uses the Motorola 68881 floating point co-processor (the Motorola 68882 is being used in later machines) which delivers in the range of 60,000–150,000 floating point operations per second (kiloflops) per node. A new floating point daughter board using the Weitek chip set has been added to each of the nodes, which boosts this performance to 2.5-10 megaflops per node. To this date the Weitek can only support 32-bit floating point operations and therefore has had limited utility for large-scale electromagnetics problems. The retrofitting of the daughter boards to 64-bit floating point arithmetic is expected to take place in the fall of 1988 when Weitek releases the new double precision chip.



Each Node:

2 MC68020 CPU's + MC68881 Co-processor

4 Mbyte DRAM

128 Kbyte Static RAM

Weitek Floating Point Accelerator

Performance:

Computation: 1-14 Mflops/node

Communication: 2.0 Mbyte/sec/channel (Synchronous)

0.5 Mbyte/sec/channel (Asynchronous)

Figure 2.1. Hardware configuration for the Mark III Hypercube

Each node of the hypercube has 4 megabytes of dynamic random access memory (DRAM). There is an additional 128 kilobytes of static RAM which can be utilized for the executable code if the code is sufficiently small (i.e., about 100 kilobytes). Codes run approximately 15% faster in the fast static RAM than in the dynamic RAM.

C. SOFTWARE

Since the hypercube has memory distributed among nodes rather than shared memory, a message passing capability has been developed to permit nodes to communicate with one another. Two styles of communication have been implemented. The first is the synchronous or crystalline operating system (CrOS). In this communicating regime nodes run asynchronously but align activity when communication is required. For some applications the message passing requirements are highly irregular and would therefore result in large idle times waiting for nodes to synchronize. For such applications the Mercury operating system was developed which allows messages to be queued in an incoming buffer and then read as needed and likewise to be queued in an outgoing buffer and sent when the message is complete. The application program can flow freely between the two styles of communication. The measured throughput rate of CrOS messages is 2 megabytes per second per channel with a node capable of communicating on all of its channels simultaneously. The throughput rate for Mercury is 0.5 megabytes per second.

The Mark III Hypercube supports both the C and FORTRAN programming languages. An operating system entitled "Time Warp" has been developed to support discrete event simulations. Relatively little effort has been applied to this date to incorporate symbolic languages on the hypercube. There has, however, been some work on a version of distributed Prolog and a Lisp interpreter.

D. NEW DEVELOPMENTS

The fact that the Mark III Hypercube continues to evolve to respond to the needs of the user community is apparent in some of the newest developments. The addition of a memory management unit (MMU) which will enable such capabilities as multi-processing and virtual memory on a node is near completion. Global buffer boards which attach a subset of the nodes to the back plane allow the user to reset a portion of the hypercube (a "sub-cube") and to load in new executable code and data to a particular sub-cube of the hypercube. Both of these capabilities become particularly important as the hypercube is expanded to larger configurations.

One of the newest developments which has recently become available is the ability to attach external devices to the nodes. A Concurrent Input/Output (CIO) interface board

was developed which interfaces the node to a VME-compatible peripheral device. This CIO board has a Motorola 68020 CPU and 2 megabytes of memory. The current board design has four communication channels, which permits it to be configured in a 3-dimensional CIO communication hypercube, leaving one channel to interface directly to an application hypercube node. Future board layouts are anticipated to include additional channels in the expectation that some applications may actually require one device per node (e.g., databases). With a CIO interface attached to each node, nodes then could access information on the disk drives via the CIO communication hypercube without using any application hypercube communication channels (Figure 2.2). A prototype CIO system became available in June, 1988 with 4 disk drives attached to a 32-node hypercube--one drive per 8-node sub-cube. The addition of disk drives to nodes is essential for the electromagnetics application. It will permit the solution of even larger problems as well as saving partial solutions for later use (see Section IV.C.5.c).

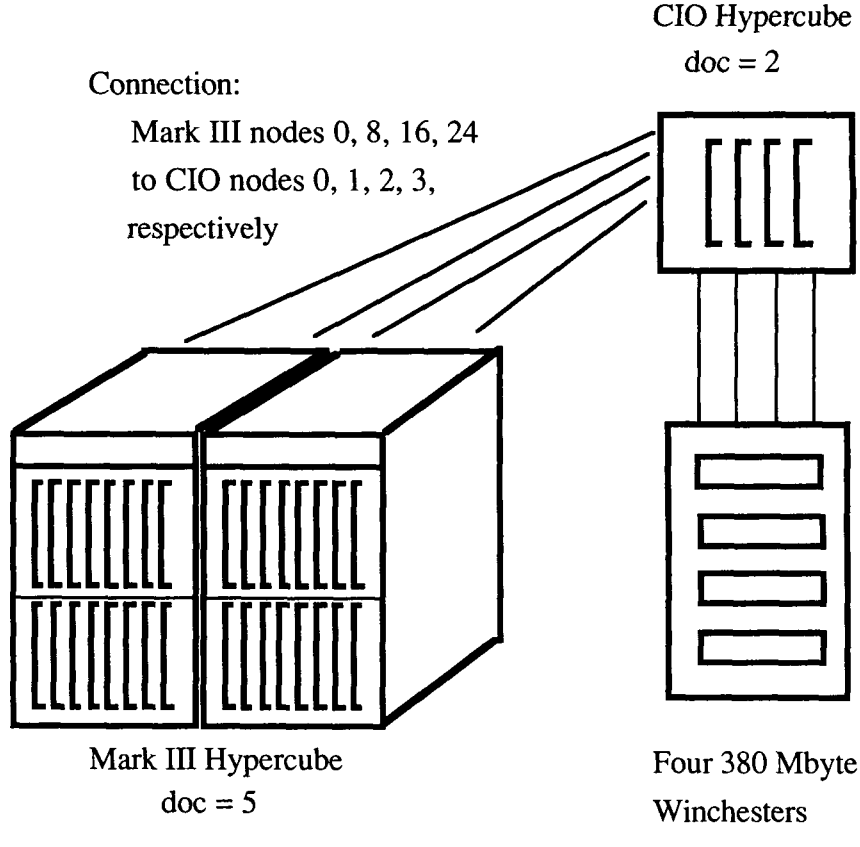


Figure 2.2. Prototype hypercube Concurrent I/O system

SECTION III

THE FINITE DIFFERENCE TIME DOMAIN CODE

A. INTRODUCTION

Electromagnetic fields interacting with a dielectric or conducting structure produce scattered electromagnetic fields. To model the fields produced by complicated, volumetric structures, the finite difference time domain (FDTD) method employs an iterative solution to Maxwell's time-dependent curl equations. Because FDTD tracks the evolution of scattered fields in time, the method naturally lends itself to problems involving transient fields. K. Yee's work on FDTD considered such applications [3-3]. A. Taflove and his colleagues at Northwestern University applied the method to the determination of steady state fields and the calculation for radar cross sections (RCSs) [3-4]. Describing the use of FDTD for calculating steady state fields and RCSs on the Mark III Hypercube is the purpose of this section.

Several sequential FDTD codes exist. Converting a FDTD code from Lawrence Livermore National Laboratory (LLNL) was our first attempt at producing a parallel FDTD code [3-1]. The modular form and straightforward coding of this program greatly eased the task of parallel decomposition. However, the LLNL staff designed the code for the analysis of transient currents. Taflove provided us with a copy of his FDTD code along with a report describing the FDTD method and its applications to RCS studies [3-2]. We added to our code the features required to make an RCS code with similar capabilities to Taflove's code's. At this point, references to the FDTD code describes our parallel implementation of the finite difference time domain method.

The goal of the FDTD code is to track the propagation of an incident electromagnetic wave into a volume of space containing a dielectric or conducting structure and observe the wave's interaction with the scattering object. Wave tracking ends when sinusoidal steady state behavior occurs at each lattice cell or after a sufficient number of cycles of the incident field. A nonfluctuating value for the monostatic RCS examined after every cycle of the incident field is a more practical convergence criterion.

FDTD uses a finite difference approximation, in rectangular coordinates, to Maxwell's curl equations to explicitly solve for the fields at the current time. This iteration scheme replaces the need for a simultaneous solution for all field components [3-3]. We envision the discretization of Maxwell's curl equations with a three-dimensional discrete lattice in space and a discrete time step.

We model the scattering object by embedding the object in the discrete lattice. The relative permittivity, ε, relative permeability, μ, electric conductivity, σ, and magnetic conductivity, σ_m mathematically describe the object. These parameters can be tensors which are functions of space. We do not consider the case where the material parameters are functions of time. We assign discrete values of these parameters to points on the discrete lattice to specify the scattering object.

The parallel implementation of FDTD divides the global lattice of discrete field components into blocks of nearly equal dimensions. The code assigns neighboring blocks to nodes directly connected by communication channels. This decomposition scheme assures that each node can perform its field updates with resident information and information communicated by neighboring nodes. This code allows us to solve problems requiring as many as 2,048,000 unit cells on a 32-node Hypercube. For smaller problems, the code produces solutions in a fraction of the time required to solve the same problems on sequential computers.

We compare the results from the parallel FDTD code using a variety of other analysis codes. We compare the reported current on the surface of a perfectly conducting cube with results from a finite difference code obtained from Taflove [3-2]. We also use the FDTD code and the Taflove code to examine the RCS from a dielectric flat plate scatterer. In the test case section of this report, Section VI, we compare the FDTD code with the Numerical Electromagnetics Code (NEC) [3-7] and a code computing the analytic solution. In this section we find the near scattered field and the bistatic RCS.

B. GENERAL THEORY

The equations governing the interaction of electromagnetic radiation with anisotropic dielectric or conducting structures are Maxwell's curl equations (3.1 and 3.2).

$$\overleftrightarrow{\mu}\begin{bmatrix}\partial_t H_x\\ \partial_t H_y\\ \partial_t H_z\end{bmatrix} + \overleftrightarrow{\sigma}_m\begin{bmatrix}H_x\\ H_y\\ H_z\end{bmatrix} = \begin{bmatrix}\partial_z E_y - \partial_y E_z\\ \partial_x E_z - \partial_z E_x\\ \partial_y E_x - \partial_x E_y\end{bmatrix} \tag{3.1}$$

$$\vec{B} = \overleftrightarrow{\mu}\,\vec{H} \qquad \vec{J}_m = \overleftrightarrow{\sigma}_m\,\vec{H}$$

$$\overleftrightarrow{\varepsilon}\begin{bmatrix}\partial_t E_x\\ \partial_t E_y\\ \partial_t E_z\end{bmatrix} + \overleftrightarrow{\sigma}\begin{bmatrix}E_x\\ E_y\\ E_z\end{bmatrix} = \begin{bmatrix}\partial_y H_z - \partial_z H_y\\ \partial_z H_x - \partial_x H_z\\ \partial_x H_y - \partial_y H_x\end{bmatrix} \tag{3.2}$$

$$\vec{D} = \overleftrightarrow{\varepsilon}\,\vec{E} \qquad \vec{J} = \overleftrightarrow{\sigma}\,\vec{E}$$

The tensors μ, σ_m, ε, and σ determine the properties of the scattering object. The FDTD code implements the special case in which these tensors only contain nonzero entries along the diagonal. $\mu(x,y,z)$ is the permeability tensor at coordinates (x,y,z). σ_m is the magnetic conductivity. ε is the permittivity. σ is the electric conductivity. B, H, J_m, D, E, and J are the usual electromagnetic field quantities: magnetic induction, magnetic field, magnetic current, electric displacement, electric field, and electric current.

From the differential equations, we construct six finite difference equations by first expressing equations 3.1 and 3.2 as separate scalar equations in rectangular coordinates. Next, we employ two-point central differencing in both spatial and temporal coordinates to discretize the scalar equations. References [3-1] and [3-2] discuss the mathematical details. The following equation 3.3 is the finite difference equation for the x component of the magnetic field. Equation 3.4 is the finite difference equation of the x component of the electric field. The finite difference equation for the other four components have a similar form.

$$\begin{aligned}&\tilde{\mu}_{11}\left(i,j+1/2,k+1/2\right) H_x^{n+1/2}\left(i,j+1/2,k+1/2\right) =\\ &\tilde{\mu}^{*}_{11}\left(i,j+1/2,k+1/2\right) H_x^{n-1/2}\left(i,j+1/2,k+1/2\right)\\ &+\frac{E_y^n\left(i,j+1/2,k+1\right) - E_y^n\left(i,j+1/2,k\right)}{\Delta z} + \frac{E_z^n\left(i,j,k+1/2\right) - E_z^n\left(i,j+1,k+1/2\right)}{\Delta y}\end{aligned} \tag{3.3}$$

$$\tilde{\mu}_{11} = \frac{\mu_{11}}{\Delta t} + \frac{\sigma_{m_{11}}}{2} \qquad \tilde{\mu}^{*}_{11} = \frac{\mu_{11}}{\Delta t} - \frac{\sigma_{m_{11}}}{2}$$

$$\tilde{\varepsilon}_{11}(i+1/2,j,k)\, E_x^n(i+1/2,j,k) = \tilde{\varepsilon}^{*}_{11}(i+1/2,j,k)\, E_x^{n-1}(i+1/2,j,k) + \frac{H_z^{n-1/2}(i+1/2,j+1/2,k) - H_z^{n-1/2}(i+1/2,j-1/2,k)}{\Delta y} + \frac{H_y^{n-1/2}(i+1/2,j,k-1/2) - H_y^{n-1/2}(i+1/2,j,k+1/2)}{\Delta z} \qquad (3.4)$$

$$\tilde{\varepsilon}_{11} = \frac{\varepsilon_{11}}{\Delta t} + \frac{\sigma_{11}}{2} \qquad \tilde{\varepsilon}^{*}_{11} = \frac{\varepsilon_{11}}{\Delta t} - \frac{\sigma_{11}}{2}$$

I, j, or k indexes the edges of unit cells. Superscripts on the field components index the discrete time steps. Δy and Δz indicate the discrete increments in the y and z directions. Δt indicates the time step increment per iteration. Other subscripted quantities are tensor elements of μ, σ_m, ε, or σ. To obtain accurate results, the size of the unit cell (Figure 3.1), the spatial finite difference step, must be a small fraction of the electrical size of the scatterer [3-2]. Once we specify the unit cell size, we determine the time step by the Courant stability condition [3-4].

FDTD constructs a lattice in coordinate space from several unit cells. The discrete electric field components, E_X, E_Y, and E_Z, lie on the edges of each unit cell. For example, E_X lies on the midpoint of edges in the x direction. The discrete magnetic field components, H_X, H_Y, and H_Z, occur on the centers of faces of each unit cell. For example, H_X lies on the center of faces perpendicular to the x direction. At discrete electric field locations, FDTD assigns permittivity and electric conductivity. At discrete magnetic field locations, FDTD assigns permeability and magnetic conductivity [3-3].

The finite difference form of Maxwell's curl equations determines the spatial and temporal update of the discrete field components on the lattice. At time $n\Delta t$, the magnetic fields, on cell faces and at half a time step back, update neighboring electric field components on cell edges. For example, FDTD uses H_Y and H_Z components to update E_X components. FDTD then increments time by half the time step, Δt . At time $(n+1/2)\Delta t$, the new electric field components on cell edges update neighboring magnetic field components on cell centers. FDTD then increments time again by half of Δt. This process continues

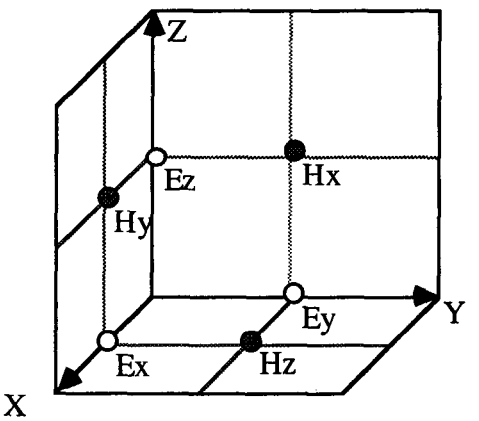


Figure 3.1. Electric and magnetic field components in a typical unit cell

until the discrete field components have constant maximum amplitudes and constant phase relative to a fixed iteration number [3-2]. Another check for convergence is nonfluctuation in the monostatic RCS after each cycle of the incident field.

Equations 3.1 and 3.2 give the relationship between total field components. Equations 3.3 and 3.4 give examples of updating total field components at each time step iteration. However, we can also write Maxwell's equations using scattered and incident electromagnetic field components. Starting with this set of equations, the finite difference algorithm would update the entire lattice in scattered fields. Before the introduction of second-order-correct radiation boundary conditions and RCS calculations, the FDTD code used the scattered-field update method. We describe radiation conditions and RCS calculations in following paragraphs. However, for several reasons, we choose to use the total-field update method near the scattering object and the scattered-field method near the truncation planes. The two update schemes require an interface region where the incident field adds to the scattered field. Taflove's report describes this update scheme in great detail [3-2]. This report best states the reasons for the use of this mixed update method: "(a) The high-dynamic-range, total-field formalism is retained for the entirety of the interacting structure, permitting accurate computations of low-level fields penetrating into cavities through apertures, and into shadow regions. (b) The scattered-field formalism is retained for the lattice truncation region, permitting a very accurate simulation of the radiation condition. (c) The incident wave contribution need be computed or stored only for the field components at the rectangular surface connecting regions 1 [total field region] and 2 [scattered field region]. This results in much less computation or storage than if the incident field were to be computed at all points within the interacting structure to implement a pure scattered field formalism. (d) The scattered near field in Region 2 can be easily integrated to derive the far-field scattering and radar cross section...."

C. RADIATION BOUNDARY CONDITIONS

The computation lattice cannot extend infinitely in all directions due to memory and computation time limitations. Therefore, we truncate the computation lattice. At the lattice boundary we cannot use the FDTD method to update field values, because the method requires points outside the lattice. Therefore, we apply a radiation condition to the scattered electric fields at the lattice truncation planes to simulate an infinite lattice. We need not apply a radiation condition to the discrete magnetic field components because these field components do not occur on the truncation planes. Formulas for first and second order radiation conditions, which contain first and second derivatives in space and time, and their corresponding central difference expressions are given in reference [3-5]. A discussion of the derivation of radiation conditions from the scalar wave equation is given in reference [3-6].

The application of second-order-correct radiation boundary conditions at the truncation planes require the scattered electric fields. However, equations 3.1 to 3.4 are total field equations. According to the Taflove method, we use an interface surface on the computation lattice to separate the region of total electric and magnetic fields and the region of scattered fields [3-2]. This surface is arbitrarily set to three cells in from the truncation planes in all directions. We also introduce the incident electromagnetic field on this interface surface. This method computes the total fields in the region of the scattering object.

Figure 3.2 illustrates the update of an x component of the scattered electric field on a truncation plane perpendicular to the y axis and with normal vector in the positive y direction. We label this point with the number 1. We cut a section perpendicular to the x axis to illustrate the update of point 1. To update point 1 at time $n\Delta t$, second order correct radiation boundary conditions need the value of the scattered electric field at points 1 and 2 at time $2\Delta t$ back and time Δt back. These conditions also require the value of the field points 3 to 10 at time Δt back.

Equation 3.5 gives the update equations for electric field points on the truncation plane shown in Figure 3.2. The variable n is the current time step index. I and k are discrete field indices in the x and z directions. Nyc is the number of cells in the y direction. Δt is the discrete time step. Δx, Δy, and Δz are the discrete spatial increments in the x, y, and z directions. Similar equations exist for the discrete x component of the electric field on truncation planes perpendicular to the z direction, for the discrete y component of the

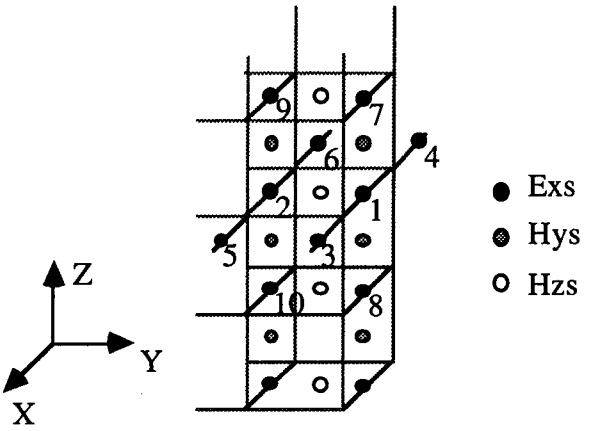


Figure 3.2. Exs points 2 to 10 update Exs point 1 on a truncation plane perpendicular to the y axis

electric field on truncation planes perpendicular to the x and z directions, and for the discrete z component of the electric field on truncation planes perpendicular to x and y directions.

$$
\begin{aligned}
E_x^n(i,nyc+1,k) &= E_x^{n-2}(i,nyc,k) \\
&+ \frac{c\,\Delta t - \Delta y}{c\,\Delta t + \Delta y}\left[E_x^n(i,nyc,k) + E_x^{n-2}(i,nyc+1,k)\right] \\
&+ \frac{2\,\Delta y}{c\,\Delta t + \Delta y}\left[E_x^{n-1}(i,nyc,k) + E_x^{n-1}(i,nyc+1,k)\right] \\
&+ \frac{\Delta y\,(c\,\Delta t)^2}{2\,(\Delta x)^2\,(c\,\Delta t + \Delta y)}\Big[E_x^{n-1}(i+1,nyc+1,k) - 2\,E_x^{n-1}(i,nyc+1,k) \\
&+ E_x^{n-1}(i-1,nyc+1,k) + E_x^{n-1}(i+1,nyc,k) - 2\,E_x^{n-1}(i,nyc,k) + E_x^{n-1}(i-1,nyc,k)\Big] \\
&+ \frac{\Delta y\,(c\,\Delta t)^2}{2\,(\Delta z)^2\,(c\,\Delta t + \Delta y)}\Big[E_x^{n-1}(i,nyc+1,k+1) - 2\,E_x^{n-1}(i,nyc+1,k) \\
&+ E_x^{n-1}(i,nyc+1,k-1) + E_x^{n-1}(i,nyc,k+1) - 2\,E_x^{n-1}(i,nyc,k) + E_x^{n-1}(i,nyc,k-1)\Big]
\end{aligned}
\tag{3.5}
$$

D. RADAR CROSS SECTION

Radar cross section (RCS) calculations require steady state far fields. However,

the FDTD code tracks the time evolution of the near fields. Therefore, the FDTD code incorporates magnitude and phase calculations for sinusoidal steady state near fields and near to far field transformations in order to calculate RCS.

The code evaluates the steady state magnitude and phase in the following manner. The FDTD algorithm marches in time through several cycles of the incident field. After several cycles, the fields reach steady state. The code then monitors the peaks and valleys of the sinusoidally varying waveform by computing the first and second time derivatives. The code monitors only the electric and magnetic fields on an integration surface. The integration surface is one cell further out than the scattered field interface surface (III.C). The code computes the magnitudes as one half the difference between peak and valley amplitudes. It computes the phase by recording the time step number at which a peak occurs and subtracting from a reference time step number. The FDTD code converts the resulting time difference to radians by multiplication with the incident field angular frequency.

The following equations describe the RCS calculations. Equation 3.6 illustrates the computation of the RCS, σ.

$$\sigma = 4\pi r^2 \frac{E_\theta^* E_\theta + E_\phi^* E_\phi}{E^{i\,*} E^i} \qquad r \to \infty \tag{3.6}$$

Equations 3.7 and 3.8 give the expressions for the far field E_θ and E_ϕ used in equation 3.6.

$$E_\theta = -ik_o \eta_o \Big(A_x \cos\theta \cos\phi + A_y \cos\theta \sin\phi - A_z \sin\theta + \eta_o^{-1} \left(-F_x \sin\phi + F_y \cos\phi \right) \Big) \tag{3.7}$$

$$E_\phi = -ik_o \eta_o \Big(-\eta_o^{-1} \left(F_x \cos\theta \cos\phi + F_y \cos\theta \sin\phi - F_z \sin\theta \right) - A_x \sin\phi + A_y \cos\phi \Big) \tag{3.8}$$

Equations 3.9 and 3.10 give the magnetic and electric vector potentials that appear in equations 3.7 and 3.8.

$$\vec{A} = \frac{e^{-ik_o r}}{4\pi r} \iint_s \left(\vec{n} \times \vec{H}^s \right) e^{ik_o r' \cos \xi} ds \tag{3.9}$$

$$\vec{F} = \frac{e^{-ik_o r}}{4\pi r} \iint_s \left(-\vec{n} \times \vec{E}^s \right) e^{ik_o r' \cos \xi} ds \tag{3.10}$$

The variables in the above equations are the following: r is the distance of the observation point, r' is the distance of the source point, k_o is the wave number of the incident field, η_o equals 376.7303 ohms, θ and ϕ are the angles of the observation point, and the E^s and H^s vectors are the complex fields obtained from magnitude and phase evaluations on the integration planes. The integration is over a cubic surface surrounding a volume containing the scatterer. The vector n is the unit outward normal on this surface.

The mathematical details of the near to far field transformations, as well as a more in-depth discussion of the second-order radiation condition and the magnitude and phase computation, are found in reference [3-2].

E. PARALLEL DECOMPOSITION

The FDTD code uses a spatial decomposition of the lattice of unit cells. The FDTD program fills the computation space with several unit cells. It then divides this global lattice into blocks of nearly equal dimensions. The FDTD code assigns neighboring blocks of the global lattice to hypercube nodes directly connected by communication channels. This decomposition scheme assures that each node can perform its discrete field updates, described in section III.B above, with resident information and information communicated by neighboring nodes.

1. Update of Discrete Field Components on the Computation Lattice

We illustrate the update of field components within the volume of the computation lattice by considering a 7 x 7 x 7 unit cell lattice. Figures 3.3 and 3.4 show the decomposition of this small lattice. In this example, the FDTD code uses four

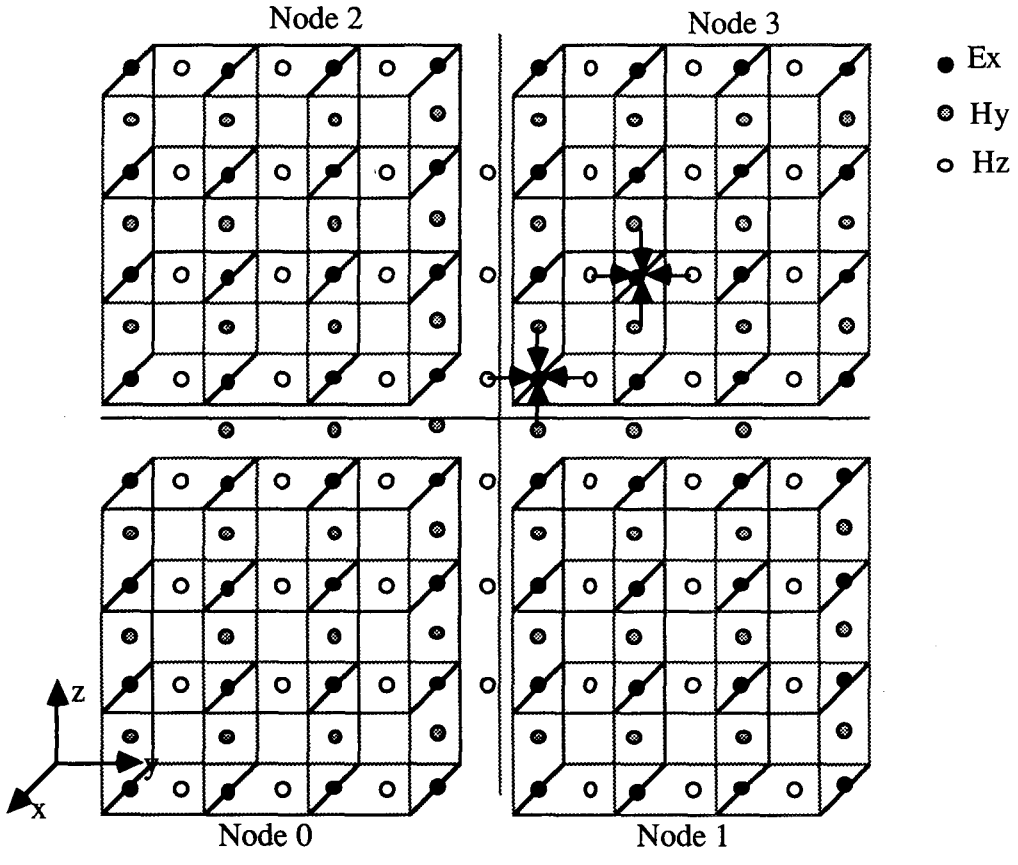


Figure 3.3. The decomposition of the global lattice among four nodes of the hypercube, in which the black arrows symbolize the magnetic field components used to update the E_x component within nodes and between the boundaries of nodes

nodes of the hypercube. It assigns a 7 x 4 x 4 cell block to node 0, a 7 x 3 x 4 cell block to node 1, a 7 x 4 x 3 cell block to node 2, and a 7 x 3 x 3 cell block to node 3. Both figures show a cut in a plane perpendicular to the x axis. In this example, there are seven such planes along the x direction. For eight active nodes, the program will divide the global lattice in the x direction. The code will assign four cells in the x direction to the nodes responsible for negative x values and three cells in the x direction to the nodes responsible for positive x values.

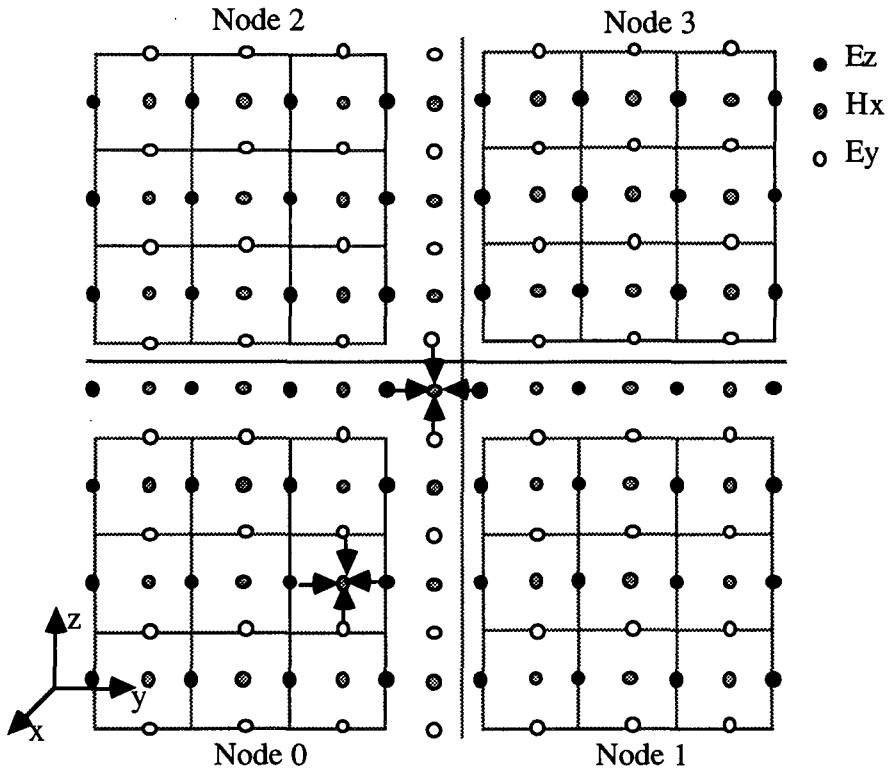


Figure 3.4. The decomposition of the global lattice among four nodes of the hypercube, in which the black arrows symbolize the electric field components used to update the H_x component within nodes and between the boundaries of nodes

Figure 3.3 illustrates the update of the x component of the electric field, E_x. To update the discrete values of E_x on the lattice, FDTD requires the value of E_x at the previous time step, the values of neighboring H_y and H_z discrete fields at half a time step back, and the values ε_{11} and σ_{11}. Refer to the finite difference equation 3.4. For an E_x component in the middle of a block assigned to a particular node, the update is the same as in the sequential case. For an E_x component on a boundary between nodes, neighboring nodes must communicate before the update. The black arrows pointing to a central E_x component illustrate the two types of updates that can occur within node 3.

The parallel FDTD code updates E_y and E_z components in a similar manner. For these components we can construct illustrations similar to Figure 3.3 for cuts perpendicular to the y and z planes.

Figure 3.4 illustrates the update of the x component of the magnetic field, H_x. To update the discrete values of H_x on the lattice, the FDTD code requires the value of H_x at the previous time step, the values of neighboring E_y and E_z discrete fields at half a time step back and the values μ_{11} and σ_{m11}. Refer to the finite difference equation 3.3. For an H_x component in the middle of a block assigned to a particular node, the update is the same as in the sequential case. For an H_x component on a boundary between nodes, neighboring nodes must communicate before the update. The black arrows pointing to a central H_x component illustrate the two types of updates that can occur within node 0.

The parallel FDTD code updates H_y and H_z components in a similar manner. For these components we can construct illustrations similar to Figure 3.4 for cuts perpendicular to the y and z planes.

2. Update of Discrete Field Components on the Truncation Planes

Second-order-correct radiation boundary conditions work well with the spatial decomposition of the global lattice. A processor responsible for field points on the truncation planes, the planes marking the boundary of the computation lattice, has enough resident and communicated field information to update truncation plane points using second-order-correct radiation conditions.

Refer back to the example illustrated in Figure 3.2. If any of the points 3 to 10 reside in neighboring nodes, the node containing point 1 must communicate with its neighbors before the update occurs.

3. Radar Cross Section Calculation

The spatial decomposition of the global lattice also works well for calculating the radar cross section, RCS. Several nodes in the hypercube contribute to the integration that yields the electric and magnetic vector potentials (equations 3.9 and 3.10). The FDTD code combines the local contributions to these potentials to calculate E_θ and E_ϕ (equations 3.7 and 3.8). These globally accumulated field values contribute to give the RCS, σ (equation 3.6).

RCS calculations depend on the magnitude and phase of discrete field components on an integration surface. The FDTD code uses a cubic surface surrounding the scattering object as the integration surface [3-2]. Portions of the cubic integration surface reside in different nodes. The program obtains the magnitude of field components on this integration surface by recording the peak and valley of the steady state sinusoidal wave forms. When a peak occurs, the program also records the time step. From this time step information, the time step increment Δt, and the angular frequency of the incident field, the program calculates the phase relative to a reference time step.

F. PERFORMANCE OF THE FDTD CODE ON THE HYPERCUBE

To analyze the performance of the FDTD code, we identify the computing intensive parts of the program. FDTD contains input/output, initialization, and setup subroutines. However, the time step iteration loop, in which the program performs field updates, radiation boundary condition updates, internode communication, and magnitude and phase tracking, comprises almost all of the execution time.

One method of efficiency measurement fixes the number of unit cells in each node while increasing the number of active nodes. If the code were 100% efficient and if the number of active nodes increases by a factor N, the execution time of the code should remain constant because the total computational load also increases by a factor N. However, the time may not remain constant because of the added internode communication. Note that, with each increase in the number of active processors, FDTD solves a different problem because the global lattice size increases.

We concentrate on the various components of the iteration loop and give results for this method of efficiency measurement.

Figure 3.5 shows four sets of timing runs. The horizontal axis indicates the number of active processors, 1, 2, 4, 8, 16, or 32. The vertical axis shows the execution time in milliseconds. Hollow squares indicate the maximum reported internode communication time per iteration. Because nodes responsible for the truncation planes of the global lattice may communicate less, nodes report distinct internode communication time. Hollow triangles indicate the maximum reported time per iteration to perform second-order-correct radiation boundary updates. Filled squares indicate the maximum reported time per iteration to perform electric and magnetic field updates within the volume of the

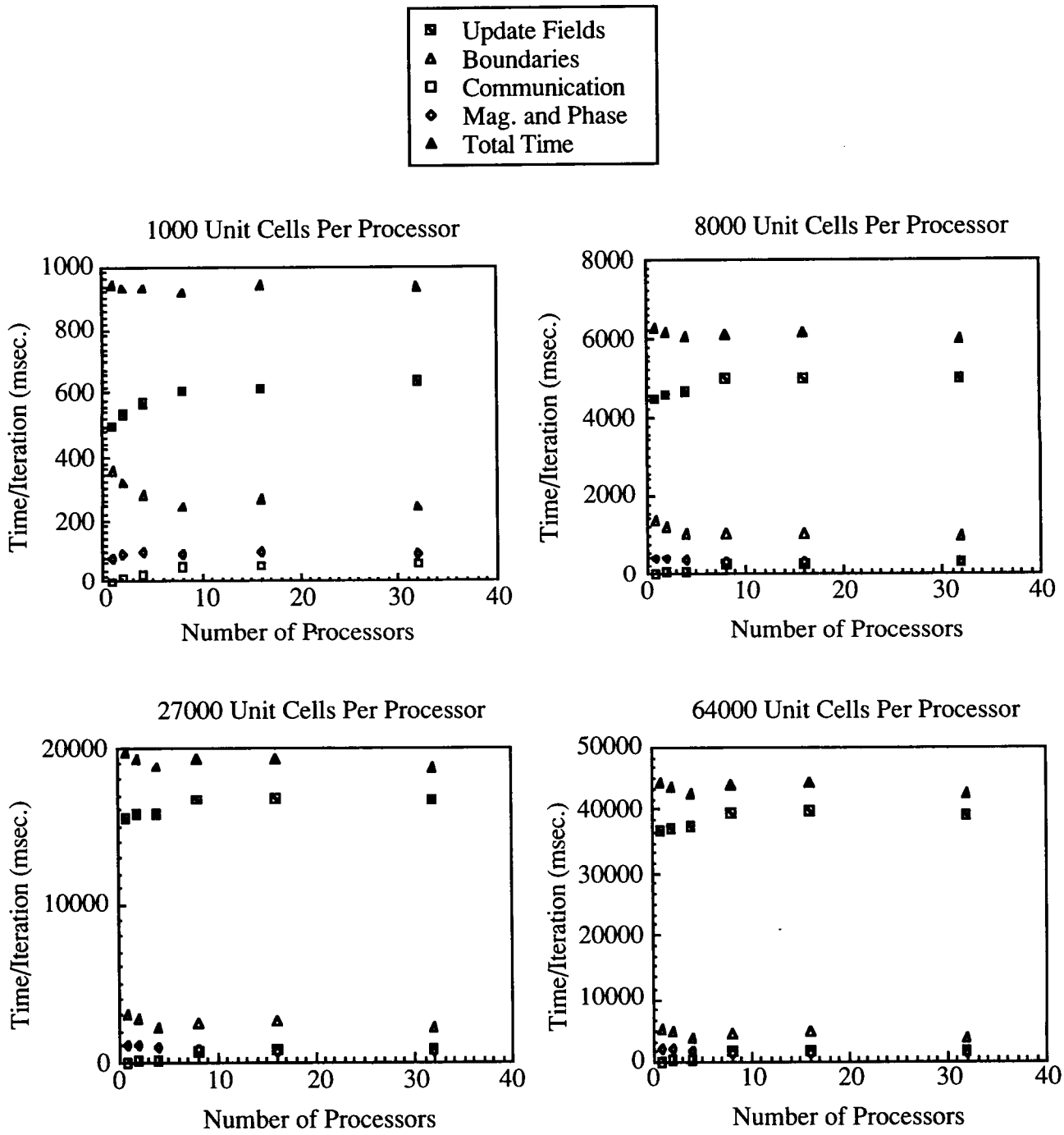


Figure 3.5. Four sets of timing runs for a fixed number of unit cells per node of the hypercube

computation lattice. Diamonds indicate the maximum reported time per iteration to perform magnitude and phase tracking on the integration surface. Lastly, filled triangles indicate the maximum reported total time per iteration. We used 1000, 8000, 27,000, and 64,000 unit cells per node.

The various components of the iteration loop predictably follow certain trends. The communication time per iteration increases with the number of processors. We expect this increase because as the number of processors increases, a given node has more neighbors with which to exchange information. However, the communication time is a small fraction of the total time per iteration and is a small fraction of the time per iteration to perform field updates.

The communication time should hit an upper limit for a given density of cells per node because a node can communicate in, at most, six directions for the three-dimensional FDTD method and because the amount of communicated information in a given direction remains constant for a fixed density. The existence of this upper limit is possible if nodes do not wait while other nodes in the configuration communicate. All nodes involved in a communication step should first write a packet of information, read the packet sent by its neighbor, and then continue to send and read additional packets. The current version of FDTD does not have this communication scheme.

As the number of nodes increases, the time per iteration to perform radiation boundary updates decreases for processor configurations of 2, 4, 8, and 16. For 32 active processors, the time per iteration increases from the time reported for 16 processors. As the number of processors increases, there are two conflicting influences on the time to perform radiation boundary updates. Although the number of unit cells remains constant in each node as we increase the number of nodes, FDTD still breaks up the truncation planes among the processors on the periphery of the global lattice. This division is the reason for the decreased times in 2, 4, 8, and 16 nodes. Second-order boundary condition updates also require internode communication. This communication is the reason for the increased time in 32 nodes.

There is a slight increase in the time per iteration for field updates as the number of processors increases. As the number of processors increases, some perform less boundary condition updates. Because the number of cells remains constant, these nodes must now perform additional field updates within the volume of the computation lattice.

For a fixed density of cells per processor, the four plots indicate that these competing times result in a faster total execution time per iteration in 4, 8, 16 and 32 nodes when compared with that for 1 and 2 nodes. Internode communication has minimal effects on the total execution time for problems saturating the capacity of each active node. With the above mentioned improvements in internode communication, this efficiency should remain high for arbitrarily large node configurations.

The parallel finite difference code performs well on the Mark III Hypercube. The efficiency for a fixed global lattice size from 1 to 32 nodes is approximately 80%. The efficiency for a fixed number of unit cells per node from 1 to 32 nodes is above 90%.

There does not exist a sequential version of the FDTD code. However, for those individuals interested in a comparison between sequential and parallel execution times for codes using the FDTD method, we compare two codes, Taflove's sequential code and the parallel FDTD code, of comparable capabilities. Taflove's sequential code is optimized because it makes no subroutine or function calls within the body of the iteration loop and explicitly uses an incident plane wave excitation. However, the parallel FDTD code makes several subroutine calls within the body of the iteration loop, including calls to functions which allow the user to specify the form of the incident field. For the conducting cube case, presented in the next section, the parallel code on the Mark III Hypercube runs approximately 22.7 times faster than Taflove's code on a VAX/750 and 8.8 times faster than Taflove's code on a VAX/785. Both VAX times are total CPU usage.

G. RESULTS FOR DIFFERENT SCATTERERS

To test the validity of the results from the FDTD code, we compare the program's reported currents on the surface of a perfectly conducting cube with the results on page 163 of reference [3-2].

We illustrate the scattering object in Figure 3.6. The scattering object is a perfectly conducting cube in vacuum. The cube is one meter in each direction. The unit cell size is 0.05 x 0.05 x 0.05 m. The computation lattice size is 2 x 2 x 2 m. The total number of cells is 40 x 40 x 40. The wave number times the length of a cube side, ks, equals 2. A plane wave is incident on the cube's front face. The direction of propagation of this plane wave is the +y direction. The electric field is polarized in the z direction with magnitude 1 V/m. The magnetic field is polarized in the x direction with magnitude

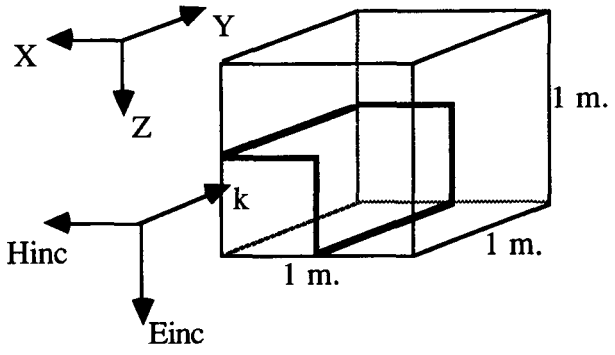


Figure 3.6. The perfectly conducting cube scatterer. The paths for evaluating the currents on the surface are bold. Arrows indicate the polarization and direction of the incident plane wave.

1/376.73 ampere/m. The frequency is 95.43 MHz. The wavelength is 3.141593 m. The number of iterations is 630 time steps with a time increment of 83.39 x 10^{-12} seconds.

In Figure 3.7, we evaluate the currents on the surface of the perfectly conducting cube scatterer. We used Taflove's sequential FDTD code and plotted the results as hollow squares. We ran the parallel FDTD code and plotted the results as plus marks (+). The top two plots show the magnitude and phase of the x component of the current on the bottom path for the perfectly conducting cube scatterer. The last two plots show the magnitude and phase of the z component of the current on the side path. The horizontal axes indicate observation points taken at intervals of 0.05 m on the current path. The vertical axes indicate either the normalized current or the phase. The Taflove and the FDTD numbers agree well.

Along with the currents, we found a monostatic radar cross section of 2.14 m^2 from the Taflove code and 2.11 m^2 from the parallel FDTD code.

We also scattered a plane wave from a anisotropic flat plate scatterer according to the parameters specified in reference [3-2]. The plate is 0.1 x 0.00625 x 0.3 m. The unit cell size is 0.00625 x 0.00625 x 0.00625 m. The computation lattice size is 0.2 x 0.1125 x 0.4 m. The total number of cells is 32 x 18 x 64. The plate has a relative permittivity equal to 1 in the z direction and 40 in the x and y directions. It has an electric conductivity of 3.72 x 10^7 mho/m in the z direction and 28 mho/m in the x and y directions. A plane wave is incident in the plus y direction. The frequency is 1.0 GHz. Each time step increments the time by 10.42 x 10^{-12} seconds.

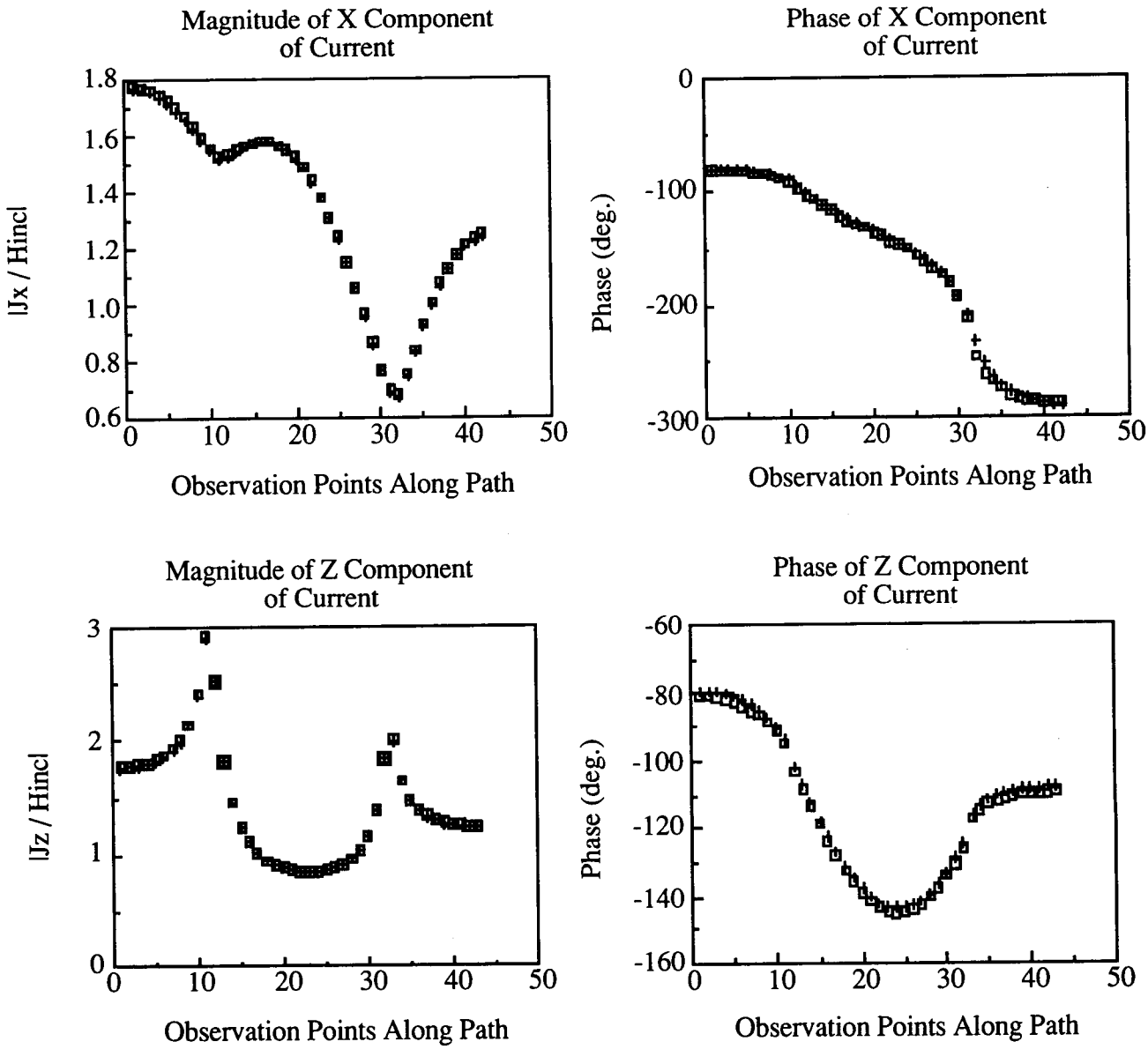


Figure 3.7. The magnitude and phase for the x component of the current on the path below the cube scatterer (top), and the magnitude and phase for the z component of the current on the side path of the cube scatterer (bottom)

We also obtained a monostatic radar cross section of 1.94 m^2 from the Taflove code at iteration 576 and 1.90 m^2 from the parallel FDTD code at iteration 576. However, according to Figure 3.8, the near fields have not completely reached steady state. This figure shows the monostatic RCS as a function of the iteration count for the flat plate scatterer.

H. CONCLUSIONS

One can code the FDTD method on the hypercube very efficiently. In the particular parallel implementation described above, communication is only between neighboring processors resulting in low communication time compared with computations occurring in the iteration loop. The large memory available on the hypercube and the ability to scale the number of hypercube nodes allow the solution of electrically large three-dimensional anisotropic scatterers.

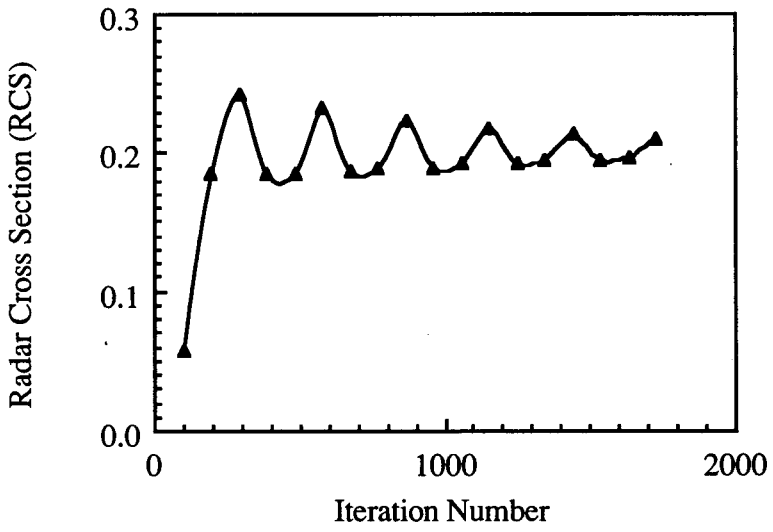


Figure 3.8. Monostatic RCS versus iteration count for the flat plate scatterer. We use this plot to check the progress of the near fields towards steady state.

REFERENCES

[3-1] K. S. Kunz, "Generalized Three-Dimensional Experimental Lightning Code (G3DXL) User's Manual," Kunz Associates, Inc., Albuquerque, NM, February 1986.

[3-2] K. R. Umashankar and A. Taflove, "Analytical Models for Electromagnetic Scattering, Part II: Finite Difference Time Domain Developments," Final Report on IITRI Project E06538, Electronics Department, IIT Research Institute, Chicago, IL, June 1984.

[3-3] K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," IEEE Trans. Antennas Prop., Vol. AP-14, May 1966, pp. 302-307.

[3-4] A. Taflove and M. E. Brodwin, "Numerical Solution of Steady-State Electromagnetic Scattering Problems Using the Time-Dependent Maxwell's Equations," IEEE Trans. Microwave Theory Tech., Vol. MTT-23, August 1975, pp. 623-630.

[3-5] G. Mur, "Absorbing Boundary Conditions for the Finite-Difference Approximation of the Time-Domain Electromagnetic-Field Equations," IEEE Trans. on Electromagnetic Compatibility, Vol. EMC-23, November 1981, pp. 377-382.

[3-6] B. Engquist and A. Majda, "Absorbing Boundary Conditions for the Numerical Simulation of Waves," Math Comp., Vol. 31, July 1977, pp. 629-651.

[3-7] G. J. Burke and A. J. Poggio, Numerical Electromagnetics Code (NEC) - Method of Moments, Lawrence Livermore National Laboratory, Livermore, CA, 1981.

SECTION IV

THE FREQUENCY DOMAIN METHOD OF MOMENTS CODE

A. INTRODUCTION

The Numerical Electromagnetics Code (NEC-2), developed at Lawrence Livermore National Laboratory (LLNL), is used for the analysis of the electromagnetic response of antennas and other metallic structures. This code computes the induced currents on structures modeled by small wires or surface patches. Other near-field quantities such as electric or magnetic fields, as well as radiated fields, are evaluated from the solution of the induced currents. The code combines an integral equation for smooth surfaces with one specialized for wires to model a wide range of structures. By using the method of moments, the integral equations are reduced to a matrix equation. The solution of the matrix equation is then used for evaluation of the currents on wire segments and surface patches. The numerical solution requires a matrix equation of increasing order as the structure size is increased relative to the wavelength of the incident field. Although there are no theoretical size limitations, modeling of structures with dimensions more than several wavelengths is impractical on conventional computers due to limitations in memory or excessive computing time. This makes the NEC program a very good candidate for conversion to the hypercube. On the hypercube, the fast parallel algorithm and large memory can extend the limits of NEC many times beyond conventional computers.

The LLNL NEC-2 is a large FORTRAN code containing 77 subroutines and many features such as:

- Computation of scattered or radiated electromagnetic fields from structures modeled by wires and surface patches
- Effects of perfect or lossy ground
- Modeling thick wires using an extended thin wire kernel
- Modeling of loaded structures (including imperfect conductors)
- Numerical Green's function
- Modeling non-radiating networks

In the parallel NEC code, all features of the sequential NEC are being incorporated except the modeling of lossy ground and non-radiating networks.

B. GENERAL THEORY AND ALGORITHMS

The detailed description of the theory of the NEC-2 method of moments is given in Reference [4-1]. In the following sections, that theory is discussed in more general terms without any attempt to cover the details.

The NEC Program uses both an electric-field integral equation and a magnetic-field integral equation to model the electromagnetic response of general metallic structures. These integral equations follow the form of an integral representation for the electric field of a volume current distribution or the magnetic field of a surface current distribution, respectively. Using these equations and the boundary condition equations on the surface results in a general integral equation where the unknowns are the longitudinal currents on wire segments and the two perpendicular components of the surface current on patches. These equations can be expressed in terms of a general linear operator as:

$$L \tilde{I} = E \tag{4-1}$$

where

L is the linear operator consisting of integral and differential operators,

I is the current on the structure, and

E represents the excitation to the system such as a voltage source or an incident field.

In the NEC program, the operator equation (4-1) is solved by a moments method in which the weighting functions, ω_i , are delta functions:

$$\omega_i (\bar{r}) = \delta (\bar{r} - \bar{r}_i) \tag{4-2}$$

where

$\bar{r}$ is the integration variable representing a point on the surface of the structure, and

$\bar{r}_i$ is the location of the center of a wire segment or a patch.

The choice of the weighting function makes this a point matching technique. The current basis functions are defined separately for wires and patches.

For M number of patches, the basis functions are simple pulse functions,

$$V_j(\bar{r}) = F_j \; I_j^p(\bar{r}), \quad j = 1, \ldots, 2M \tag{4-3}$$

where

$$I_j^p(\bar{r}) = 1 \text{ for } \bar{r} \text{ on patch j and zero otherwise.}$$

The total current on patches can be expressed as

$$\tilde{I}^p(\bar{r}) = \sum_{j=1}^{2M} V_j(\bar{r}) = \sum_{j=1}^{2M} F_j \; I_j^p(\bar{r}) \tag{4-4}$$

For N number of wires, the basis function is expanded into three terms: a constant, a sine, and a cosine. For segment j the basis function is represented by:

$$I_j^w(\bar{r}) = a_j + b_j \sin k(\bar{r} - \bar{r}_j) + c_j \cos k(\bar{r} - \bar{r}_j), \quad j = 1, \ldots, N \tag{4-5}$$

where

$\bar{r}_j$ is the location of the center of segment j,
k is the free space wave number, and
a_j, b_j, and c_j are constants.

It is assumed that this basis function has a peak on segment j and goes to zero at the other end of the segments connected to segment j (Figure 4.1).

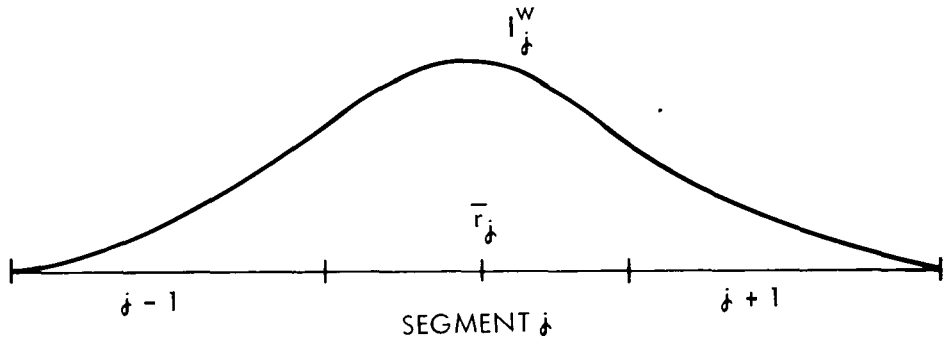


Figure 4.1. The j^{th} basis function for wires

Using the local continuity condition of charge and current, two of the three constants in each basis function are eliminated and the total current can be expressed as:

$$\tilde{I}^w(\bar{r}) = \sum_{j=1}^{2M} F_j \; I_j^w(\bar{r}) \tag{4-6}$$

where

F_j represents the remaining constants.

Combining the current expansion for patches and wires we get:

$$\tilde{I}(\bar{r}) = \sum_{j=1}^{N+2M} F_j \; I_j(\bar{r}) \tag{4-7}$$

where

I_j represents either I_j^W or I_j^P.

A set of equations is obtained by taking the inner product of equation (4-1) with the set of weighting functions in equation (4-2), ω_i :

$$\left\langle \omega_i, L\tilde{I} \right\rangle = \left\langle \omega_i, E \right\rangle \tag{4-8}$$

Due to the linearity of the operator L, substitution for I yields:

$$\sum_{j=1}^{N+2M} F_j \left\langle \omega_i , L I_j \right\rangle = \left\langle \omega_i , E \right\rangle \tag{4-9}$$

In matrix form equation (4-9) can be written as:

$$[A]\ [F] = [E] \tag{4-10}$$

where

[A] in general is a (N+2M) * (N+2M) matrix called the "interaction matrix,"

[F] is a (N+2M) * 1 array of basis function amplitudes,

[E] is a (N+2M) *1 array of excitation at the center of wire segments and patches.

In the parallel NEC program, the matrix A is factored by one of two techniques: 1) the Householder transformation, which results in an upper triangular matrix, or 2) Gaussian elimination, which results in a lower/upper pair of triangular matrices. Depending on the factorization method selected, either back substitution (for Householder transformation) or both back and forward substitution (for Gaussian elimination) are used to compute the amplitudes of the basis functions.

For structures having N wire segments and M surface patches, equation (4-10) can be written as:

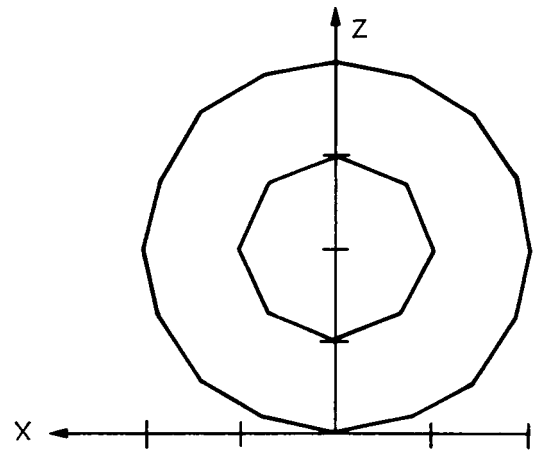
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} F_w \\ F_p \end{bmatrix} = \begin{bmatrix} E_w \\ H_p \end{bmatrix} \tag{4-11}$$

where

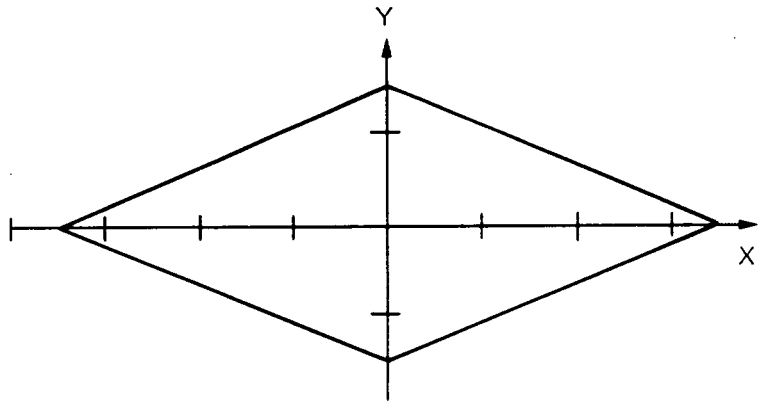
F_w and F_p are basis function amplitudes for wires and patches, respectively.

E_w and H_p are the electric fields at the center of wire segments and the magnetic fields at the center of surface patches, respectively.

The interaction matrix is now divided into 4 submatrices a, b, c, and d. A matrix element a_{ij} in submatrix a represents the electric field at the center of segment i due to the j^{th} segment's basis function centered on segment j. A matrix element d_{kl} in submatrix d represents a tangential magnetic field component at patch k due to a surface-current pulse on patch l. Matrix elements in submatrices b and c represent electric fields due to surface-current pulses and magnetic fields due to segment basis functions, respectively.



(a)



(b)

Figure 4.2. Two examples of symmetry: (a) coaxial rings (cylindrical symmetry); (b) a rhombic antenna (plane symmetry)

For problems where the structure to be analyzed has cylindrical symmetry, or planes of symmetry such as the one shown in Figure 4.2, the computation time can be reduced significantly.

In a case where there are l symmetric cells in the structure, equation 4-10 can be written as:

$$\begin{bmatrix} A_1 & A_2 & A_3 & A_{l-1} & A_l \\ A_l & A_1 & A_2 & A_{l-2} & A_{l-1} \\ A_{l-1} & A_l & A_1 & A_{l-3} & A_{l-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_2 & A_3 & A_4 & A_l & A_1 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_l \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \\ \vdots \\ E_l \end{bmatrix} \qquad (4\text{-}12)$$

where now each submatrix A_i is of dimension NPEQ = (N'+ 2M'), where N' and M' are the number of wire segments and surface patches in each symmetric cell. This would reduce the time to fill the interaction matrix by a factor of $1/l$. It can also be shown that by taking discrete Fourier transforms of equation (4-12), it can be solved by factoring l matrices of order NPEQ instead of one matrix of order l *NPEQ. This would reduce the factor time by $1/l$ and the solution (forward and back substitution or back substitution, depending on the factoring technique used) time by about $1/l$. The time to compute Fourier transforms is generally small compared to the time for matrix operations. Symmetry also reduces the number of locations required for matrix storage by $1/l$ since only the first row of submatrices needs to be stored.

In the NEC-2 program, the current of each wire segment can be approximated either by a thin wire kernel or by an extended thin wire kernel. In the thin wire approximation, the segment current is represented by a current filament. This limits the use of the thin wire models to

$$\frac{\Delta}{a} > 8$$

where

Δ = the length of each segment, and

a = the radius of each segment.

With the extended thin wire approximation, the segment current is represented by a current tube which can be used for models with

$$\frac{\Delta}{a} > 2$$

Another important feature of the NEC-2 program is its capability to model structures with lumped or distributed loads, which includes modeling with lossy wires. This is accomplished by a simple modification to the boundary condition equation. For unloaded structures, the general boundary equation is:

$$\hat{n}(\bar{r}) * \left[\bar{E}^{S}(\bar{r}) + \bar{E}^{I}(\bar{r})\right] = 0 \tag{4-13}$$

where

$\hat{n}$ is the normal vector to the structure, and

E^S and E^I are the scattered and incident fields, respectively.

For loaded structures, this equation is modified as follows:

$$\hat{n}(\bar{r}) * \left[\bar{E}^{S}(\bar{r}) + \bar{E}^{I}(\bar{r})\right] = Z_s(\bar{r}) \left[\hat{n}(\bar{r}) * J_s(\bar{r})\right] \tag{4-14}$$

where

Z_S = the surface impedance, and

J_S = the surface current density.

C. PARALLEL DECOMPOSITION OF NEC

1. Structure of the Code

As discussed in Section IV.A, the purpose of the NEC code is to calculate the radiation pattern of an object modeled by wires and patches. The user specifies the incident excitation, which will be an incident electromagnetic wave for scattering problems and a voltage source in the object for antenna problems.

The NEC code solves for currents I induced in the object by the excitation E. To do this, the unknown currents I are expanded into known basis functions with unknown amplitudes. The vector F is found by solving the matrix equation A*F=E, where E is the excitation vector and A is the interaction matrix. After the matrix equation is solved for F, the currents I induced in the object are calculated from F and the basis functions. The

radiation pattern is then calculated from the induced currents I. The structure of the sequential VAX code is illustrated in Figure 4.3.

In the NEC code developed for the Mark III Hypercube, while the basic approach outlined above is unchanged, the actual code structure is similar to the sequential VAX code. Most of the calculations are done in the hypercube elements (nodes), i.e., the fill of the interaction matrix, the factorization, and the solution. The calculation of the induced currents and the near and far fields are performed in the Control Processor (CP). These field calculations are currently being added to the element code.

The basic structure of the parallel NEC code is shown in Figure 4.4. The structure of the code in the CP is shown on the left and that of the hypercube code on the right.

a. Input. The input section of the code is performed in the CP. This section includes reading the various input data cards from a file, creating the wires and patches from the geometry input cards, setting up parameters regarding the excitation vector, and determining what output is to be calculated from the input cards.

First, the input data relating to setting up and solving the matrix equation is read and processed. This includes all of the geometry and excitation information. If the scattered radiation pattern is to be calculated for an electromagnetic wave incident from several different angles, NEC must solve the matrix equation A*F=E for the same A matrix but several different E vectors, corresponding to the different angles of incidence. In this case, the parameters needed for filling all of these E's are calculated at the start in the CP in the new subroutine ETMFILL. The information needed for filling the A matrix and the E vectors is then passed to the hypercube by communication between the CP subroutine CPCOMM and the hypercube main program NECELT. At this point, the CP is idle until it receives the solution vector F for the first excitation vector E.

b. Matrix Fill. After receiving the initialization data from the CP, the element code first calls the subroutine CMSETN, which handles the fill of the interaction matrix A. The parallel fill of this matrix is described in Section IV.C.2 below. If there is symmetry in the modeled object, the user can specify the symmetry option and, by doing so, significantly reduce the order of the matrix equation to be factored and solved.

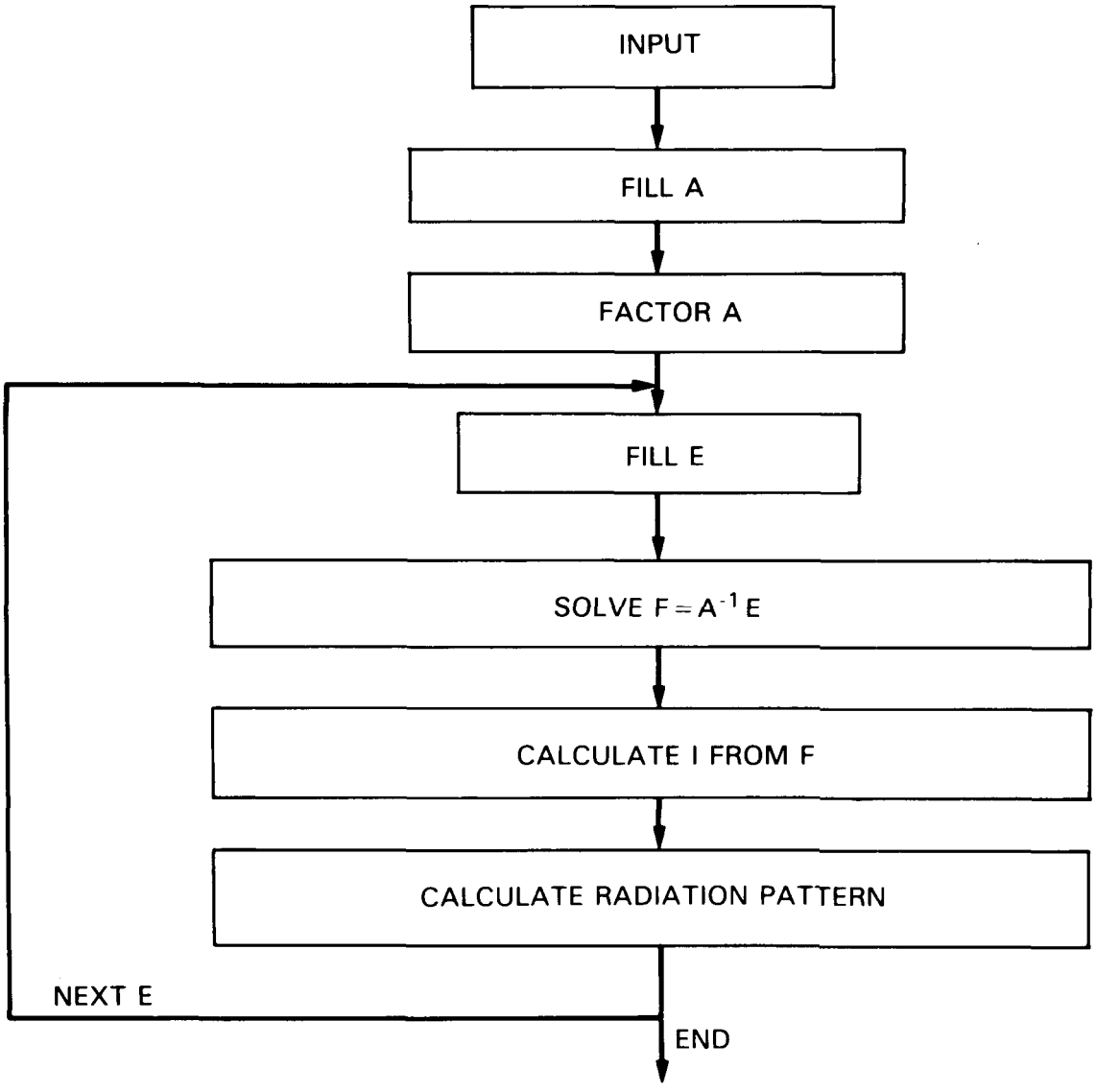


Figure 4.3. Structure of the sequential NEC-2 program

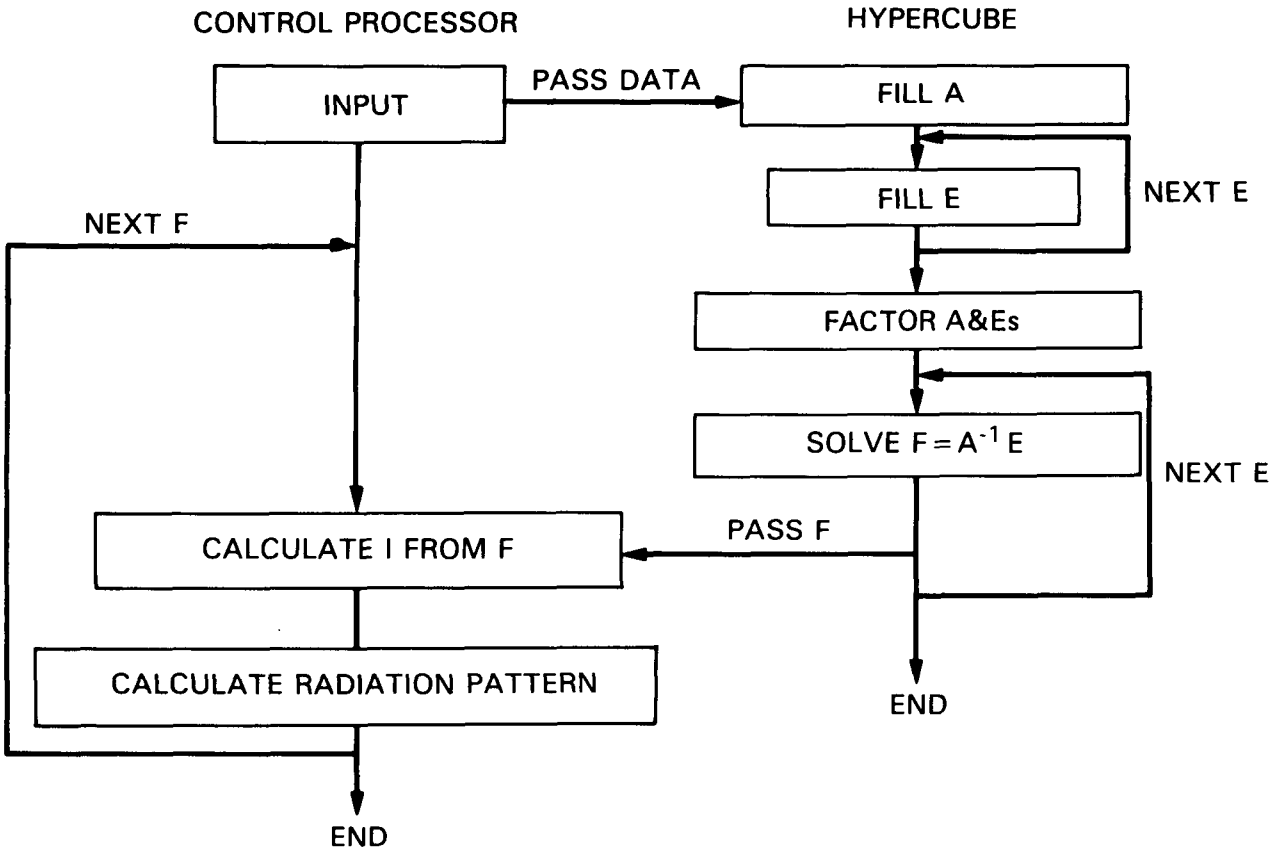


Figure 4.4. Structure of the parallel NEC program

c. Excitation Fill. After the matrix fill, the excitation E is calculated in the subroutine ELTETM, described in Section IV.C.3. If the matrix equation is to be solved for multiple right-hand sides, all E vectors are filled at this time through multiple calls to this subroutine.

For cases with l symmetric subsections, the l-point discrete Fourier transform of the E's is next performed by the subroutine ELTDFT. Multiple calls to this subroutine are made for the case of multiple excitation vectors.

The E's and their discrete Fourier transforms are calculated here before the factorization. The Gaussian elimination method of factorization does not transform the E vector(s), but the Householder transformation technique does.

d. Factorization. The main element program NECELT calls the subroutine FACTRSN, which then performs the factorization of A. For cases with l-fold symmetry, the factorization of A is done separately on the l symmetric sections. The subroutine FACTRSN makes l calls to either the Householder transformation subroutine, HHNFACTR, or the Gaussian elimination subroutine, GAUSS. For multiple right-hand sides, the factorization of A or each of A's symmetric subsections is done only once.

e. Matrix Equation Solution. After the factorization is complete, the matrix equation is solved by back substitution in the case of the Householder transformation method, or by both forward and back substitution for Gaussian elimination. The solution is done in the subroutine PARSOLV, which solves the the matrix equation separately for each of the l symmetry subsections. PARSOLV is called by the subroutine ELTSOLN. For symmetric cases, ELTSOLN performs the inverse discrete Fourier transform of the solution vector F.

After the solution vector F is found for a particular excitation E, this vector is passed back to the CP. For cases with multiple right-hand sides, the hypercube then makes another call to PARSOLV to solve the transformed matrix equation for the next E while the CP is processing the previous solution vector F. After the hypercube finds and passes back the solution vector F for the last excitation E, the hypercube code terminates.

f. Current Calculation and Output. When the CP receives its first solution vector F, the induced currents I are calculated sequentially from F by the

subroutine CABC. The CP then calculates whatever output was specified in the input data, e.g., near fields and far field radiation patterns, and writes the output to a file.

For cases with multiple right-hand sides, the CP then is idle until it receives the next solution vector F. When the output from all excitations has been found, the CP code terminates.

2. Interaction Matrix Fill

The interaction matrix A is shown in terms of its four submatrices a, b, c, and d in equation (4-11). In the NEC program the elements of these submatrices are computed by subroutine CMWW (wire-to-wire interaction), CMSW (surface-to-wire interaction), CMWS (wire-to-surface interaction), and CMSS (surface-to-surface interaction), respectively. An element A of the interaction matrix is computed by evaluating the electric field or the magnetic field at the center of an observation segment (a segment can be a wire segment or a surface patch) due to the basis function centered on another segment called the *source segment.* Figure 4.5 graphically depicts this process for a wire-to-wire interaction.

Computation of each element of the interaction matrix is therefore accomplished in three steps:

1) Defining a source segment (identified by index j) and computing some data related to that segment

2) Defining an observation segment (identified by index i) and computing data related to that segment

3) Finally, evaluating either the electric field or the magnetic field using the information obtained in the previous steps.

The order in which the first two steps are taken differs depending on the submatrix that is being filled.

In the sequential NEC program, the interaction matrix is filled inside a double DO-loop with index variables i and j, where i and j go from 1 to N (the total number of segments). Figure 4.6 show a block diagram for the sequential code. In the first DO-loop

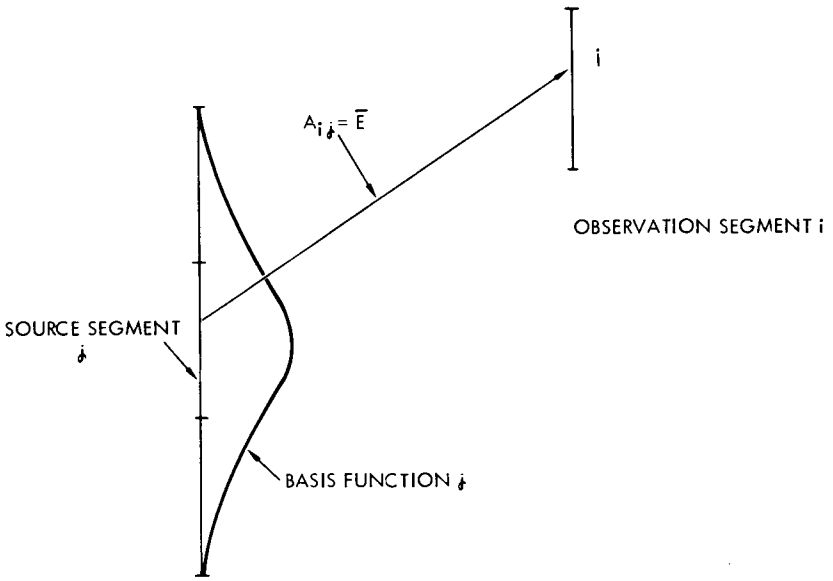


Figure 4.5. Interaction of wire segments i and j

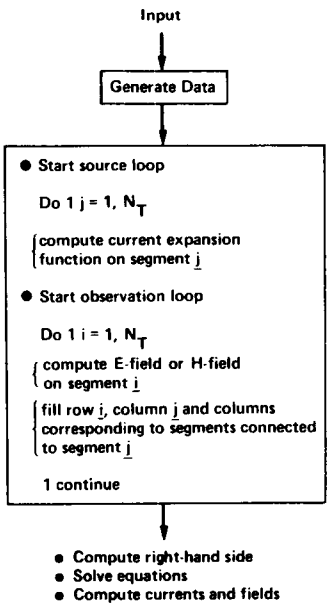


Figure 4.6. Sequential interaction matrix fill

a source segment j is picked and the current expansion function on that segment and adjacent segments are computed. In the second DO-loop an observation segment i is picked and the electric field or magnetic field on that segment is computed depending on if it is a wire segment or a surface patch. Consequently, in row i, column j and columns corresponding to segments connected to segment j are filled.

In the parallel NEC program several nodes are used to fill the interaction matrix. Therefore, each processor is responsible for filling a part of the interaction matrix. To minimize communication between nodes and redundant storage of data, each node fills full rows of the interaction matrix. The rows are assigned to the hypercube nodes according to the chart shown in Figure 4.7, that is, for a hypercube with n_p nodes, node n will compute rows n, $n+n_p$, $n+2*n_p$, and so on. With this distribution scheme, if the number of rows is equal to a multiple of the number of nodes, all nodes will fill the same number of rows. Otherwise, some nodes will fill one more row than others.

Two parallel codes are developed for the matrix fill. In the first code, called the Source Loop Sequential Code (SLSC), only the observation loop is computed in parallel. The data in the source loop is computed in the same way as in the sequential NEC program and therefore some redundant information is processed in the hypercube, which contributes to a reduction in the parallel code's efficiency. On the other hand, with this algorithm there is no need for the hypercube nodes to communicate with each other, which helps to improve the efficiency. A block diagram of the code with the sequential source loop is shown in Figure 4.8. Here, the program in each node is basically the same as in the sequential code except that in the observation loop the index i takes on different values depending on the node in which the code is running.

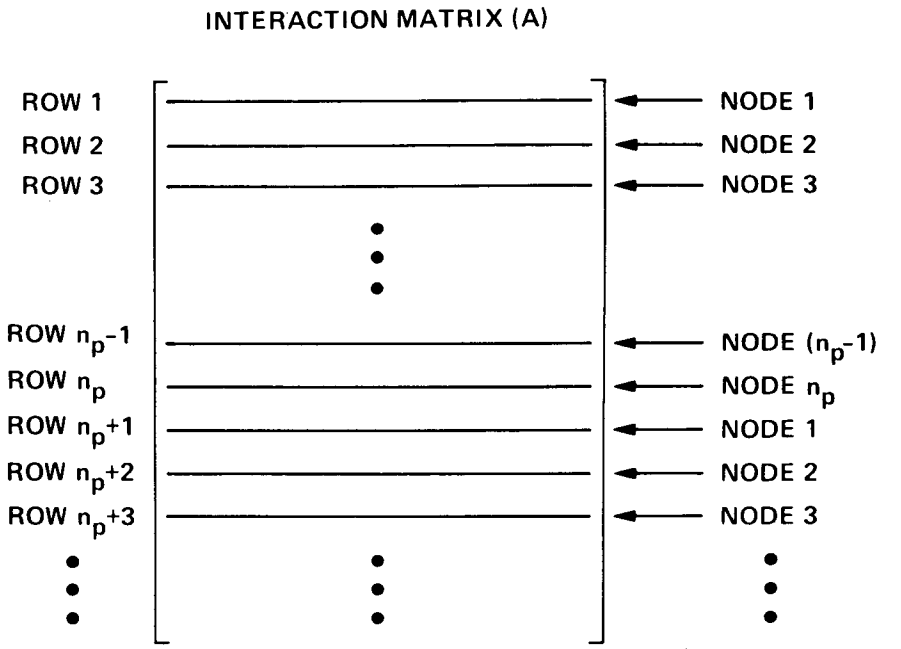


Figure 4.7. Assignment of rows to hypercube processors

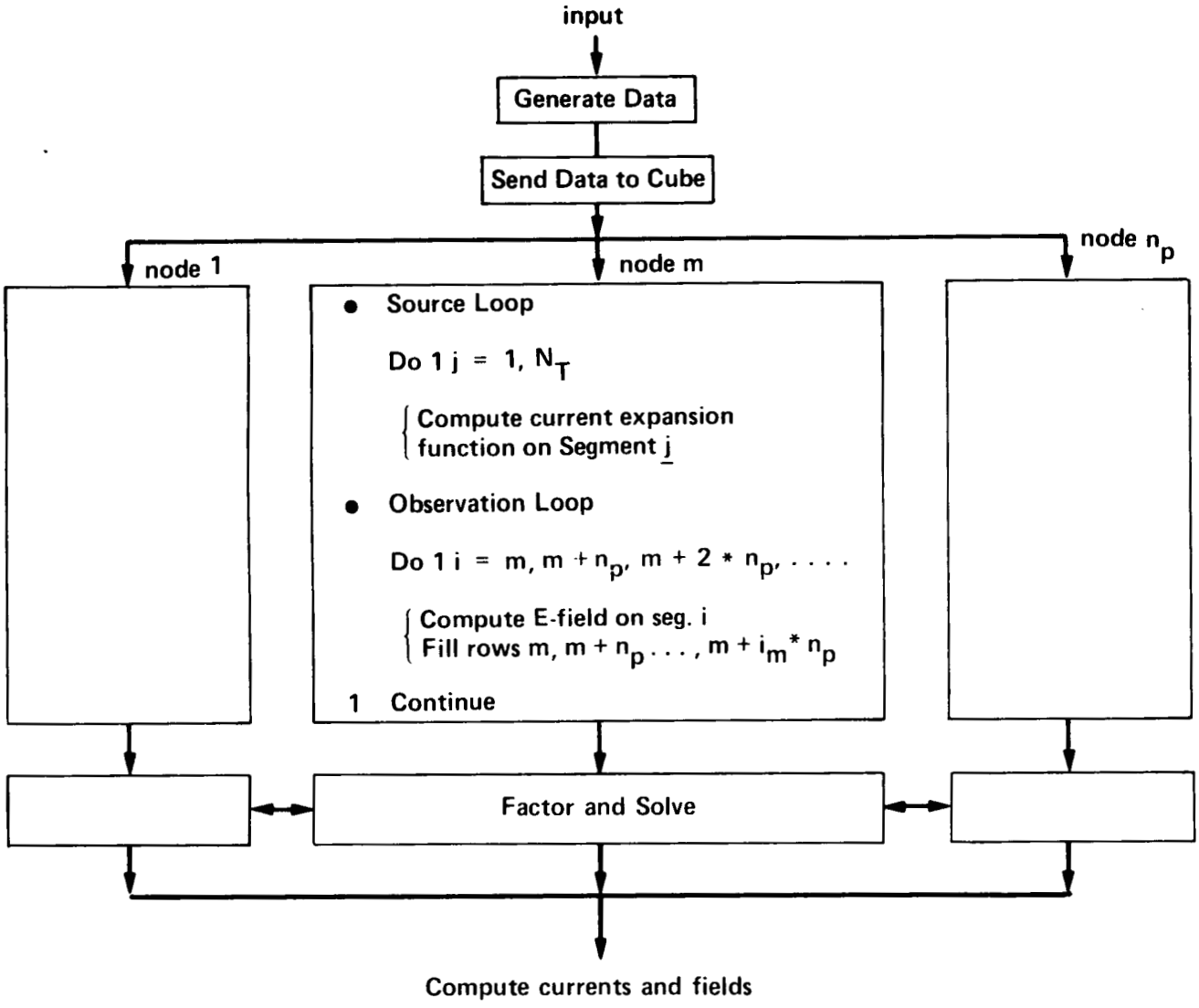


Figure 4.8. Sequential source loop for the parallel interaction matrix fill program

The second code, called Source Loop Parallel Code (SLPC), is an extension of the source loop sequential program. A block diagram of this code is shown in Figure 4.9. Here, the observation loop is computed as for the SLSC, but to avoid processing redundant information, the source loop, as well as the observation loop, is made parallel. This is implemented only for wires because the source information for patches can be computed faster than it can be communicated at all times. In this program, each node will compute the source data for segments j=m, m+n_p, m+2*n_p , and so forth. Therefore, the data for all source segments is available but divided among all of the nodes. Hence, for a node to obtain the source data for all segments, it has to receive some of that data through communication channels from other nodes. For this purpose, the nodes are mapped into a one-dimensional periodic lattice (that is, a ring) (Figure 4.10). With this arrangement, each node can send or receive information from its two neighboring nodes only. The information flow is set to be counterclockwise and therefore node m will receive data from node m-1 but sends data to node m+1.

To better describe the source loop parallel algorithm, consider the example shown on Figure 4.11. It is assumed that a problem with 6 segments is being computed by a 4-node hypercube. Now, we will observe node 1 at different times after the source loop is started at time t_0:

- At time t_1, node 1 computes source data for segment j=1.
- At time t_2, it sends data for j=1 to node 2 and receives data for j=4 from node 4.
- At time t_3, it sends data for j=4 to node 2 and receives data for j=3 from node 4.
- At time t_4, it sends data for j=3 to node 2 and receives data for j=2 from node 4.
- At time t_5, all nodes, including node 1, compute the source data for segments j=5 and j=6.

As in the case above, if there is a remainder in the division of the number of segments by the number of nodes, the data for the remaining segments is computed sequentially.

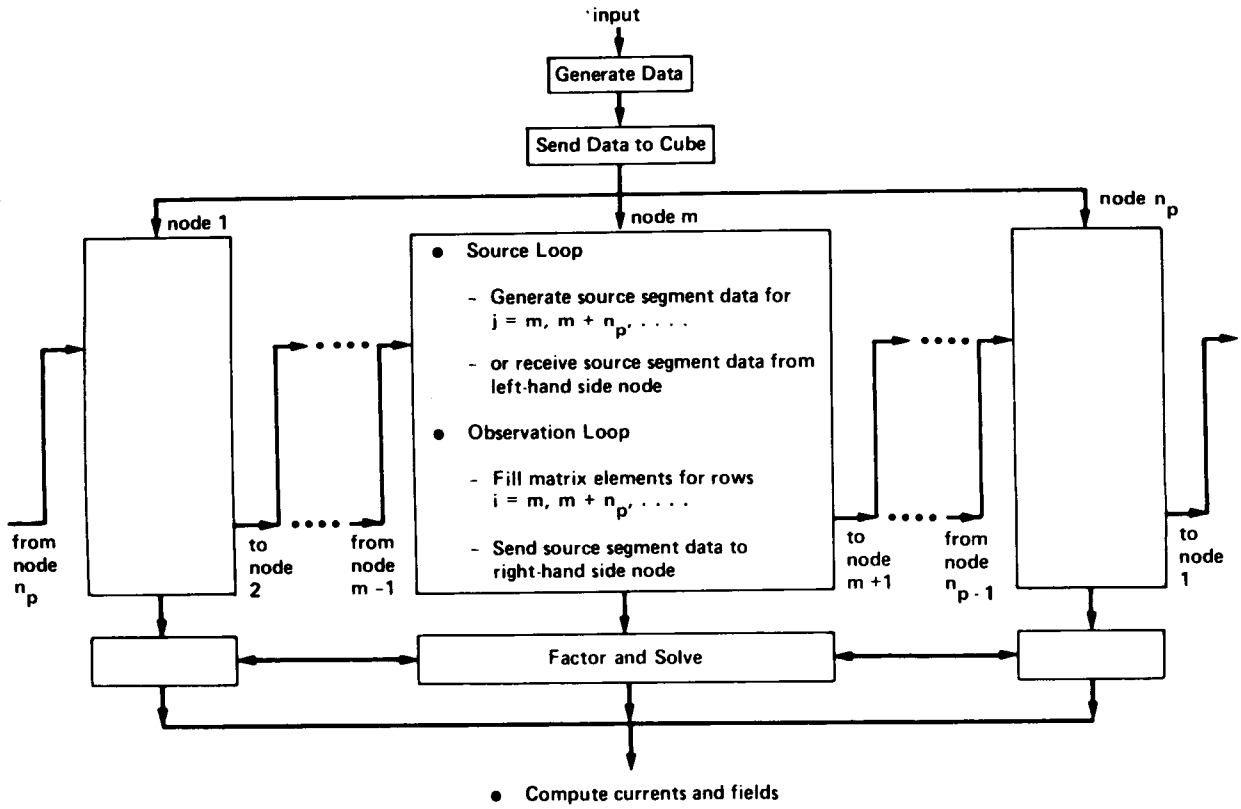


Figure 4.9. Parallel source loop for the parallel interaction matrix fill program

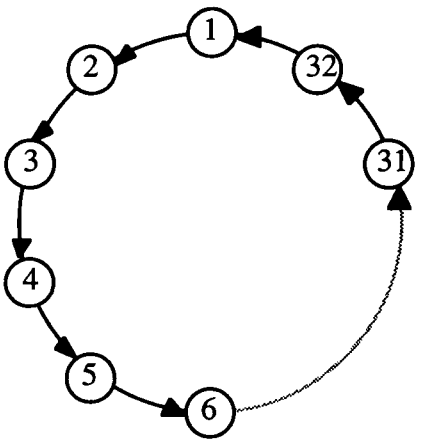


Figure 4.10. Node arrangement for the source loop parallel program in a 32-node hypercube

3. Parallel Fill of the Excitation Vector

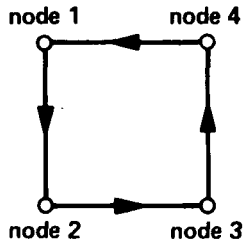
NEC solves the matrix equation A*F=E, where the excitation vector E and the solution vector F have $N_T = N+2*M$ elements for an object modeled with N wires and M patches. For objects with no symmetry, the interaction matrix A is $N_T * N_T$. For an object with l symmetric subsections, A has N columns and NPEQ rows, where $NPEQ=N_T/l$ as discussed in Section IV.4. When the solution vector F is found, it is written over the E vector and so the parallel decomposition of E also determines the parallel decomposition of F.

The parallel decomposition of the E fill is determined by the parallel decomposition of the matrix fill. For cases with no symmetry, the only restriction on E is that the i^{th} element of E be in the same node as the i^{th} row of A so that the solution algorithm can proceed without the necessity of fetching the needed element of E from a different node. Thus, the elements of E are "dealt out" to the nodes as the rows of A were (see Section IV.C.2): the first element goes to the first node, the second element to the second node, and so on, returning to the first node after the n_p elements have been dealt. The deal proceeds until all elements of E have been distributed.

For cases with symmetry, the distribution of E changes somewhat, although the parallel decomposition is still determined by the decomposition of A and the requirements of the solution algorithm. As discussed in Section IV.C, the code now solves l matrix

- Assume number of segments to be:

  N = 6 ; J = 1, 6



- Source segment data is either computed in nodes or transferred to nodes in the following sequence in node $\begin{Bmatrix}1\\2\\3\\4\end{Bmatrix}$
  - Time t_0 start source loop
  - Time t_1 compute source segment data for J = $\begin{Bmatrix}1\\2\\3\\4\end{Bmatrix}$
  - Time t_2 send data for J = $\begin{Bmatrix}1\\2\\3\\4\end{Bmatrix}$ to node $\begin{Bmatrix}2\\3\\4\\1\end{Bmatrix}$ Receive data for J = $\begin{Bmatrix}4\\1\\2\\3\end{Bmatrix}$ from node $\begin{Bmatrix}4\\1\\2\\3\end{Bmatrix}$
  - Time t_3 send data for J = $\begin{Bmatrix}4\\1\\2\\3\end{Bmatrix}$ to node $\begin{Bmatrix}2\\3\\4\\1\end{Bmatrix}$ Receive data for J = $\begin{Bmatrix}3\\4\\1\\2\end{Bmatrix}$ from node $\begin{Bmatrix}4\\1\\2\\3\end{Bmatrix}$
  - Time t_4 send data for J = $\begin{Bmatrix}3\\4\\1\\2\end{Bmatrix}$ to node $\begin{Bmatrix}2\\3\\4\\1\end{Bmatrix}$ Receive data for J = $\begin{Bmatrix}2\\3\\4\\1\end{Bmatrix}$ from node $\begin{Bmatrix}4\\1\\2\\3\end{Bmatrix}$
  - Time t_5 compute source segment data for J = 5 and 6

Figure 4.11. Example of a parallel source loop algorithm on 4 nodes

equations of order NPEQ=N_T /l , where l is the number of symmetric submatrices of A. In these cases, E is also composed of l symmetric subvectors of length NPEQ. The NPEQ rows of A have been distributed as in the case with no symmetry. Now, the l symmetric subvectors are distributed separately, always assigning the first element of the subvector to the first node, the second element of the subvector to the second node, etc., until the NPEQ elements have been distributed. In the end, the corresponding elements of each subvector of E are in the same node: elements 2, 2+NPEQ, 2+2*NPEQ are in the second node. The number of elements in each processor, then, is always a multiple of l . Figure 4.12 shows an example of decomposition of E for N segments and 3-fold symmetry.

For cases with symmetry, E, as well as the interaction matrix A, is expanded into a discrete Fourier series immediately after it is filled. At the solution time the elements of A, E, and F are the coefficients of this expansion. Although the parallel decomposition of E is determined by the decomposition of A, this decomposition has the added feature that all of the elements of E which are needed to perform the discrete Fourier transform have been assigned to the same node. Thus, no internode communication was necessary to perform the discrete Fourier transform of E and, likewise, to perform the inverse discrete Fourier transform of the solution vector F.

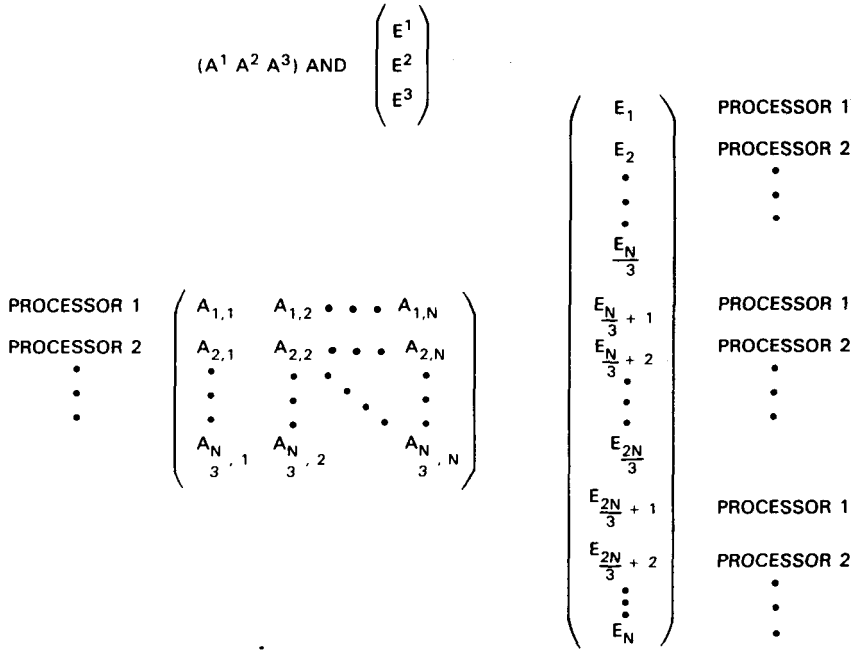


Figure 4.12. Assignment of processors for an object with 3-fold symmetry and N segments

4. Factorization and Solution

a. Householder Transformation. The Householder transformation is one of the two techniques used for the factorization of the interaction matrix [4-2]. This factorization method is well-suited to a parallel implementation because it operates on columns of the matrix where each column's set of computations is independent of the computations performed in the other columns. As the factorization progresses, the matrix is transformed into an upper right triangular matrix by a series of orthogonal transformations (Figure 4.13). The right-hand-side excitation vectors must undergo the same series of transformations at the time that the interaction matrix is transformed. In this way there is no need to store (or, for that matter, even explicitly compute) the transformation matrix.

Because the Householder transformation algorithm works on columns of the A matrix, the data must be redistributed to the nodes by columns following the matrix fill. The Householder transformation consists of NPEQ transformations (where NPEQ is the number of rows in the interaction matrix). For the first transformation, the data in the first column is distributed to all nodes using a BROADCAST call. Each node uses this column to determine the new elements for its columns of what now becomes the "working matrix" as well as to determine its elements of the new factored matrix. At the conclusion of the first transformation, the interaction matrix contains new columns for columns two through column NPEQ of the working matrix; column one will no longer be active. In the second

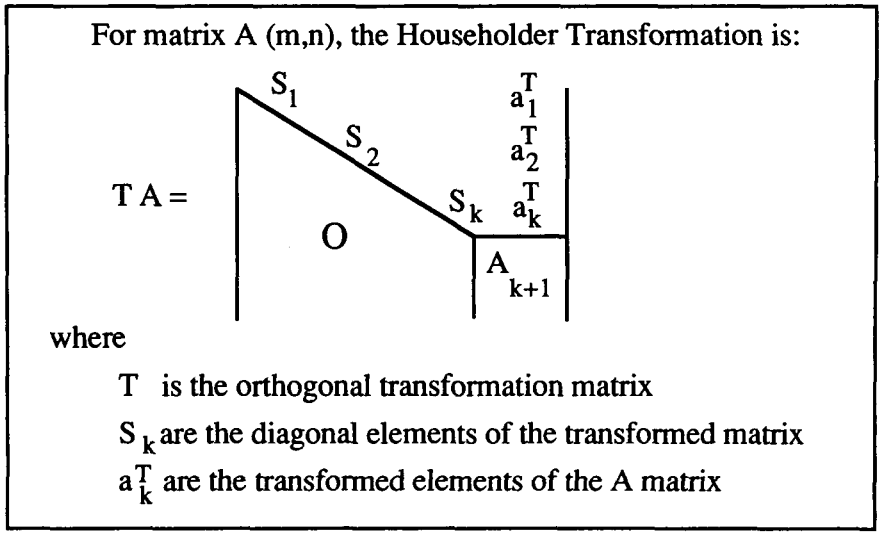


Figure 4.13. Householder transformation

transformation, the node which has column number two distributes it to all of the other nodes. Again each node computes the new elements for its active columns of the working matrix and its elements of the second row of the factored matrix. The transformation progresses, each time with one less active column, until NPEQ transformations have been performed and the complete upper right triangular matrix is constructed (Figure 4.14).

In order to economize on storage, the newly computed factored matrix row elements overwrite the inactive portion of the working matrix (Figure 4.15). Since the elements are distributed among the nodes at the end of a transformation step, the data is combined and assembled in the node which will eventually do the solution for that row. This row assignment matches the earlier row assignment in CMSETN. At the end of each transformation, the newly calculated transformed elements of the excitation vector(s) are also replaced in the source nodes, again matching the earlier row assignments. At the conclusion of the factorization subroutine, HHNFACTR, the data is in position for the back substitution.

b. Gaussian Elimination. The second technique used for factorization is a row variant of Gaussian elimination with partial pivoting (Figure 4.16) [Ref 4-3]. This factorization technique produces upper and lower triangular matrices which must be solved using both forward and back substitution. With this algorithm the interaction matrix is distributed to the nodes by rows. Each node processes those rows for which it is responsible.

The first task in each elimination step is to determine in which row (and therefore in which node) resides the maximum element (Figure 4.17). The maximum is first determined locally within each node. Then the nodes exchange maximum elements to determine which element is the global maximum. If the node which is responsible for that elimination step row does not have the maximum element, it must then exchange that row for the row which contains the maximum element. This process is called "partial pivoting." In the process of the exchange, which is completed via a global BROADCAST command, all nodes obtain a copy of the elimination row. This elimination row will be used as a multiplicand by all nodes as they next complete the remainder of the elimination step, now locally.

Performed in parallel:

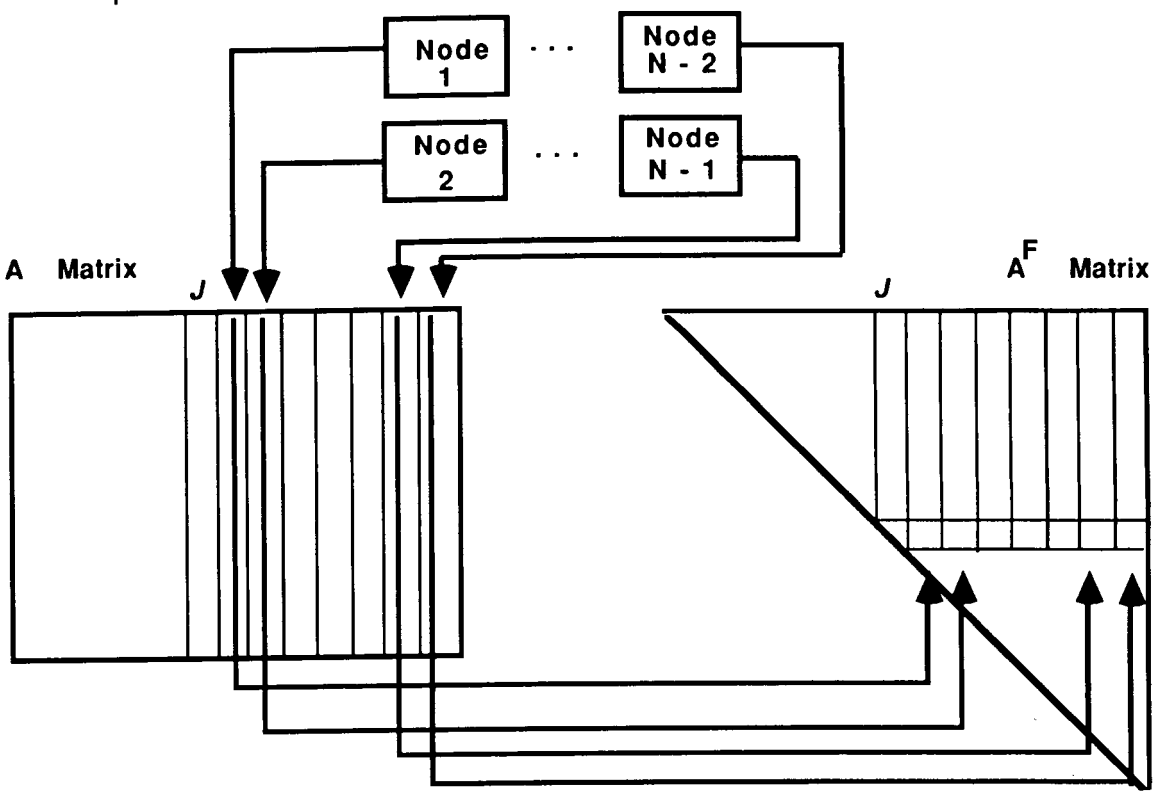
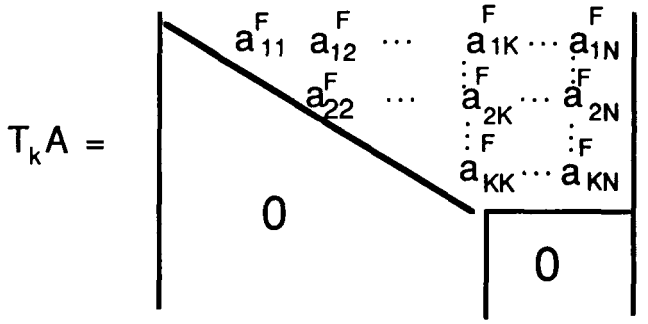


Figure 4.14. Upper right triangular matrix formed using the Householder transformation

The factored matrix A^F after the kth Householder Transformation:



To minimize storage, the A matrix columns are stored as rows. Then as factoring progresses the A^T and A^F are merged:

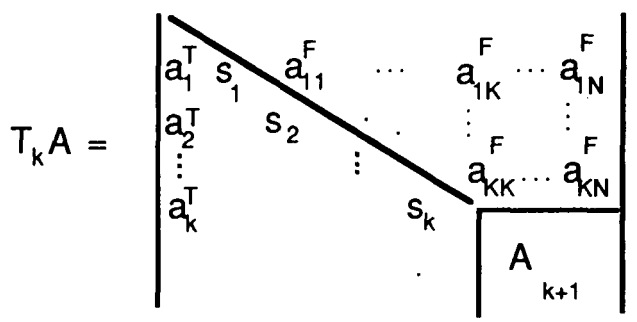


Figure 4.15. Householder transformation after the k^{th} transformation. A^T is the working matrix which is discarded after the factorization. A^F is the newly factored upper right triangular matrix.

Although both factorization techniques are implemented and the user is given the option of selecting the technique to be used as an input parameter, the Gaussian elimination technique is the one generally chosen. The Gaussian elimination factorization currently runs a factor of about 4 times faster than the Householder transformation. It is conjectured that this may be due to the additional data reorganization that the Householder transformation technique requires. Both techniques are inherently order N^3 operations. Further work is required to better understand the difference in the performance of the two algorithms. The relative performance statistics may change some when the NEC code runs using the Weitek daughter board floating point accelerator.

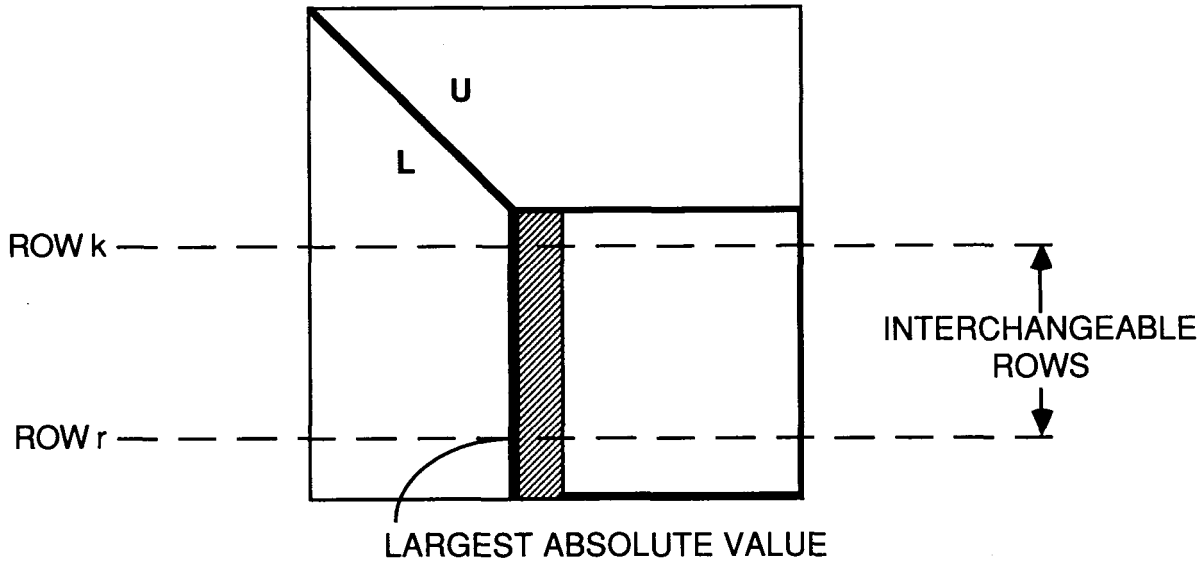


Figure 4.16. Matrix factorization by Gaussian elimination

c. Solution. The solution method depends on the method of factorization selected: for Householder transformation there is a back substitution for each right-hand-side excitation vector, and for Gaussian elimination there is both forward and back substitution for each excitation. Each node calculates the solution elements for the rows it is assigned. At the conclusion of the calculation of each row, the solution elements are broadcast to the other nodes. These solutions become multiplicands for subsequent steps of the solution.

d. Numerical Green's Function. The factored interaction matrix may be stored in a file and used in subsequent solutions by taking advantage of the Numerical Green's Function (NGF). It is as though the free space Green's function in NEC is replaced by the Green's function for the scattering object which has been stored in a file. The NGF is used to model large structures where various components will be added as an iterative process in the course of the analysis. NGF is also useful when analyzing an object

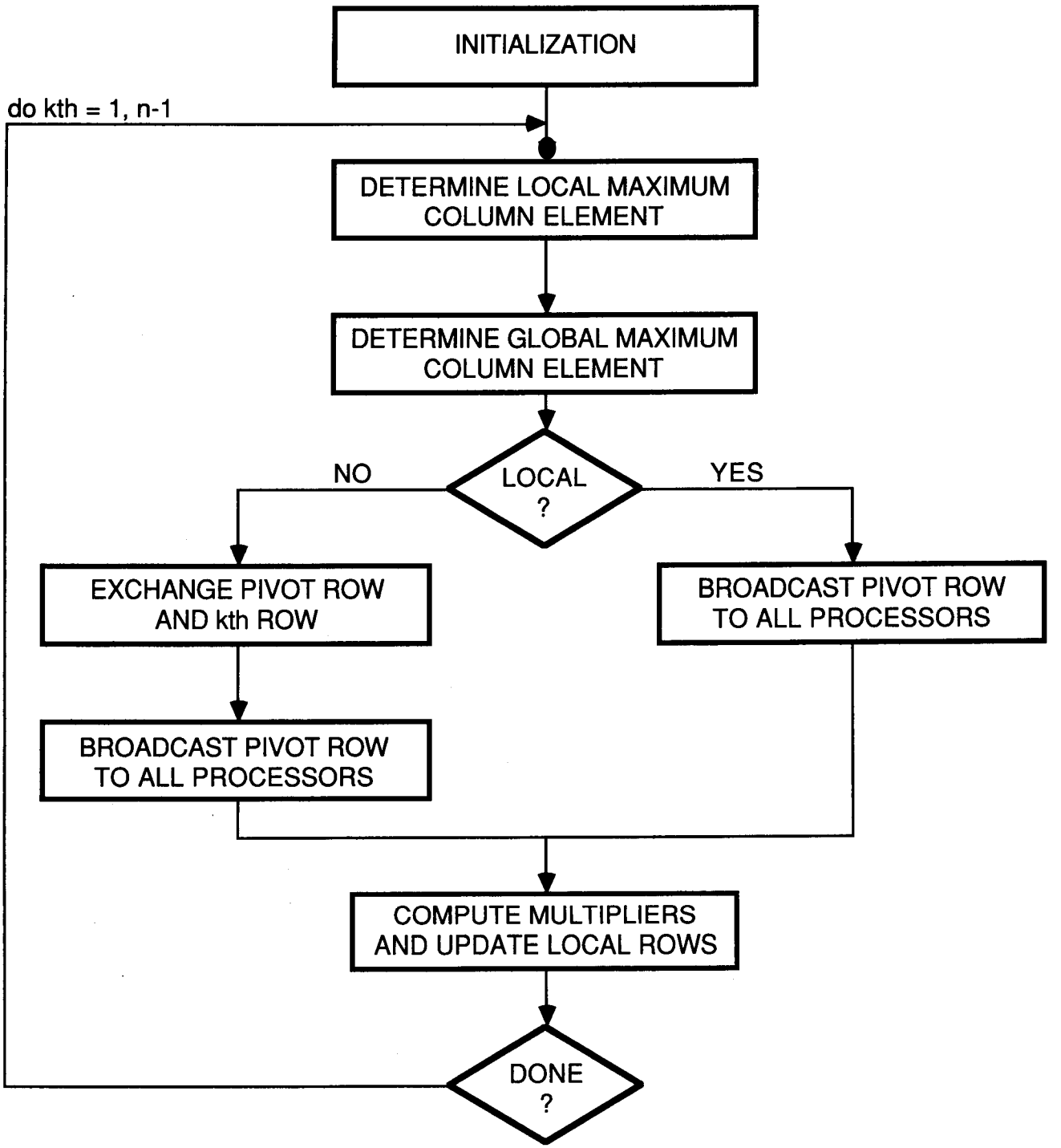


Figure 4.17. Parallel factorization by Gaussian elimination

with partial symmetry. The symmetric portion can be solved on the initial run of NEC, thereby reducing the order of the matrix to be solved and therefore the computation time. The unsymmetric portions are then added on later runs.

As implemented in NEC, the NGF does not permit incremental building of a structure. That is, all additions are recalculated in subsequent runs. Therefore, as the object grows, more and more computations are repeated. We are currently developing the capability to incrementally design an object by incorporating the new portion into the factored interaction matrix. Two techniques are being considered: a block Gaussian elimination algorithm and a matrix update by means of the Householder transformation. A third method is being investigated for cases where the number of modifications/additions are small relative to the overall structure. It is an iterative technique using the factored interaction matrix as a preconditioner.

The development of the iterative NGF capability in NEC is particularly important because the implementation of the Concurrent Input/Output (CIO) system is now completed for the Mark III Hypercube. Currently there are four 300-megabyte disk drives attached to the 32-node hypercube, one drive per 8-node sub-cube. Four more disk drives will be added in the fall of 1988. The factored matrix is stored on disks in a distributed fashion by issuing a CIOWRITE command and restored to memory from the disks by a CIOREAD.

There are three main branches to the parallel NEC code (Figure 4.18). In the normal mode, an execution flag, NGFLG, is set to *zero*, indicating that no NGF file will be read or written. If a user wishes to preserve the factored interaction matrix, the NGFLG is set to *one* and at the conclusion of the run the matrix and ancillary data are stored on the distributed disk drives. If a user wishes to modify an existing structure, then the NGFLG is set to *two*, which instructs the program to retrieve, from the disk drives, the factored interaction matrix and ancillary data from a previous run.

D. NUMERICAL RESULTS

The numerical results obtained from the parallel NEC have been checked with the results from the original sequential NEC-2 running on a VAX 11/750 and have shown excellent agreement. The results for three problems, where wires or patches, either separate or combined, are used, are discussed here. Also, examples are shown for

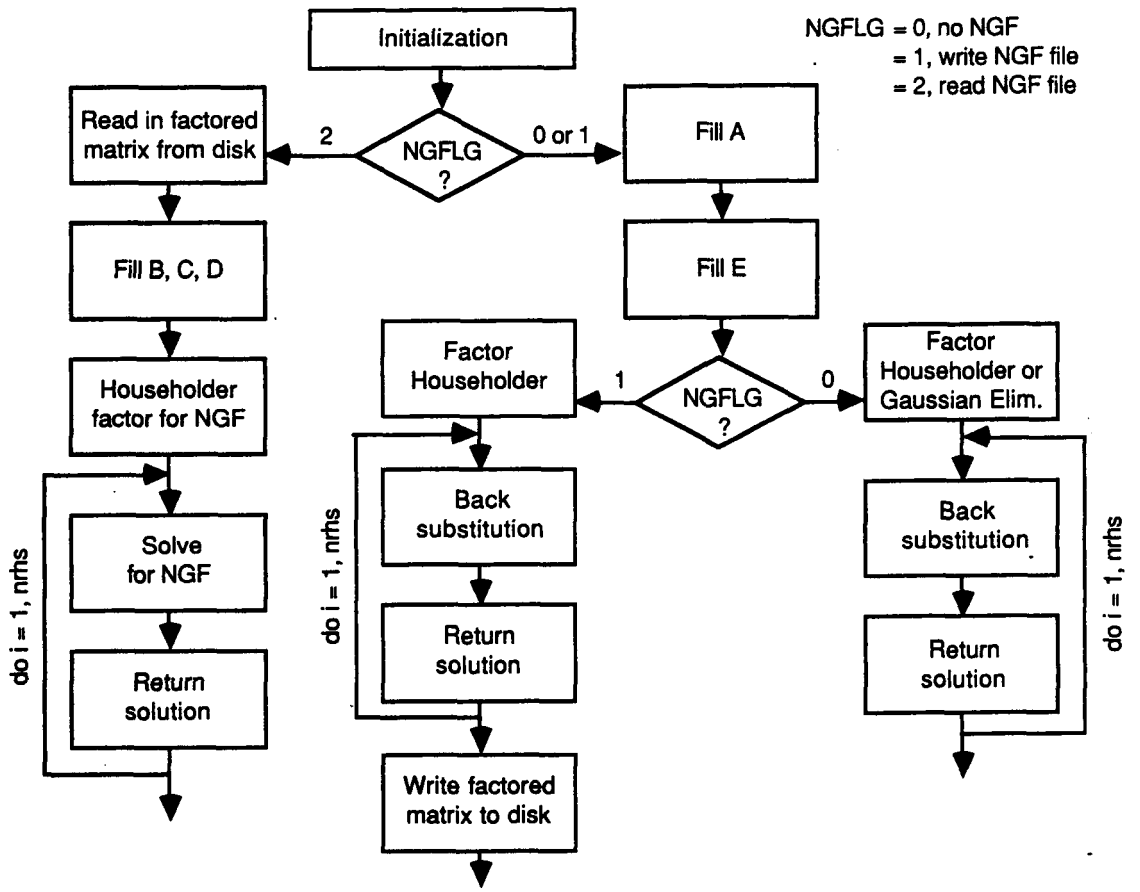


Figure 4.18. Three main branches to the parallel NEC code

validation of the code when the extended thin wire approximations or loading are employed.

1. Monopole on a Pedestal Over Perfect Ground

This problem is modeled by as many as 290 wire segments. The pedestal is modeled by several radial 2-wire sections (or ribs) as shown in Figure 4.19. Each wire, in turn, is divided into several segments. A quarter wave length monopole is placed at the center of the pedestal and is fed by a voltage source at the bottom. The far field radiation pattern of this monopole, in plane $\phi = 0$, is shown, where ϕ and θ are the standard azimuthal and polar angles in the spherical coordinate system. The solid line is from the sequential code, while the dots show the results from the parallel code. To the precision of the respective machines, excellent agreement exists between the two codes. The bend in the pattern which can be noted is due to the presence of the pedestal. By increasing the number of ribs in the wire model, the effect of the pedestal will become more significant.

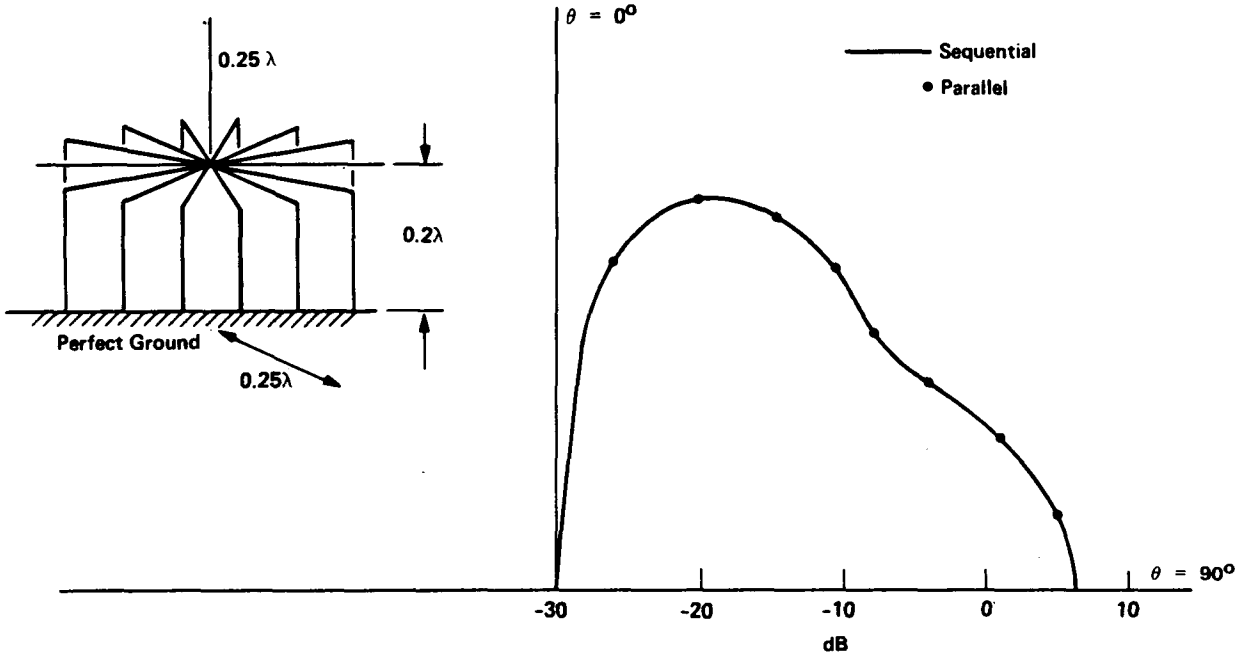


Figure 4.19. Radiation pattern of a quarter wave monopole on a pedestal over perfect ground; $\phi = 0$ cut. The structure is modeled by 130 wire segments.

2. Scattering of a Plane Wave by a Conducting Sphere

The conducting sphere is modeled by patches only. In the example shown in Figure 4.20, the sphere is modeled by 80 patches without taking advantage of the symmetry. The incident field is a uniform plane wave traveling in the -x direction and is polarized in the -z direction. The far field scattering pattern of the sphere, for the plane $\phi = 0$, is shown. The solid line shows the results from the sequential NEC code, while the dots show the results from the parallel code. As in the previous example, very good agreement exists. For this example ka = 3.0, where

$$k = \frac{2\pi}{\lambda},$$

λ = the plane wave wavelength, and

a = the radius of the sphere.

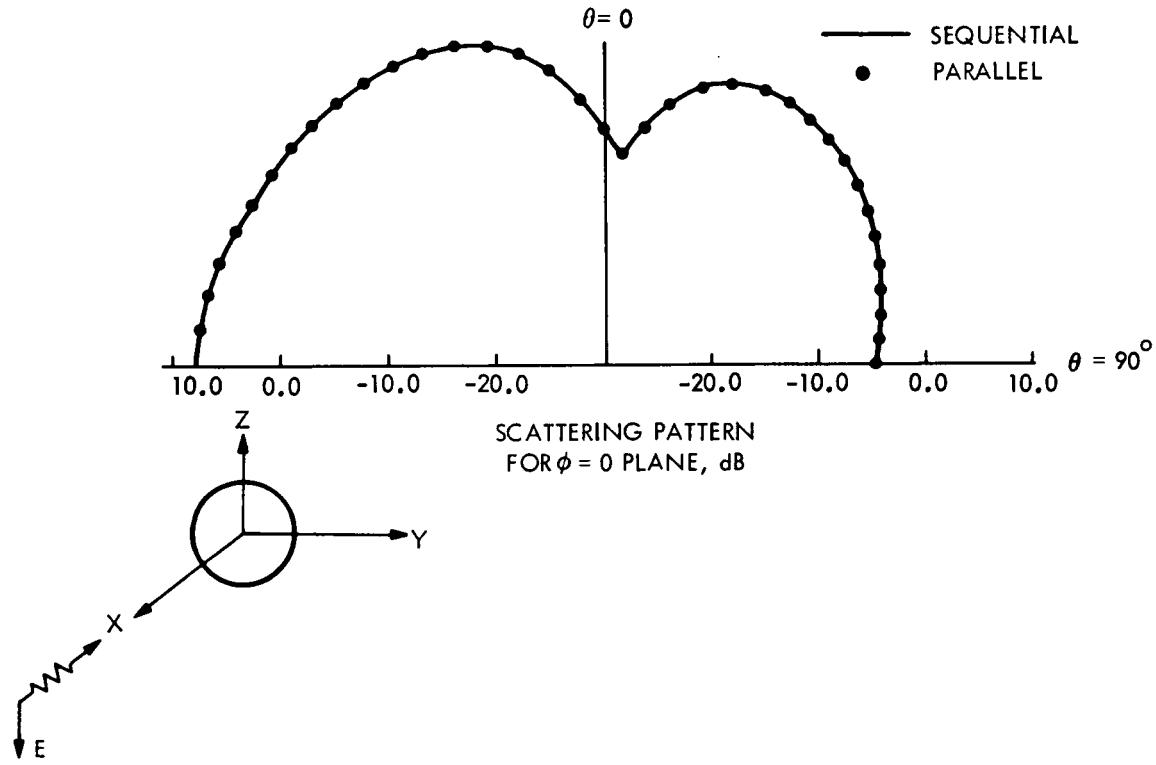


Figure 4.20. Scattering pattern of a sphere with ka = 3.0 in the $\phi = 0$ plane. The incident field is a plane wave traveling in -x direction. The sphere is modeled by 80 patches.

3. T Antenna on a Conducting Box Over Perfect Ground

This problem is modeled by both patches and wires. The box is modeled by 12 patches and the T antenna by 8 wire segments. This is the same as in example 4 in the NEC manual [4-1]. The T antenna is fed by a voltage source at the bottom. The radiation pattern of this antenna, for the $\phi = 0$ plane, is shown in Figure 4.21. The solid line shows the results from the sequential NEC and the dots show the solution from the parallel NEC.

4. Extended Thin Wire and Loading

To demonstrate the effect of using the extended thin wire kernel and loading, the monopole on a pedestal structure of Section IV.D.1 is employed. The structure is modeled by 70 segments. The radiation pattern of the monopole, for the $\phi = 0$ plane, is shown in Figure 4.22. Here, the solid line shows the results when thin perfectly

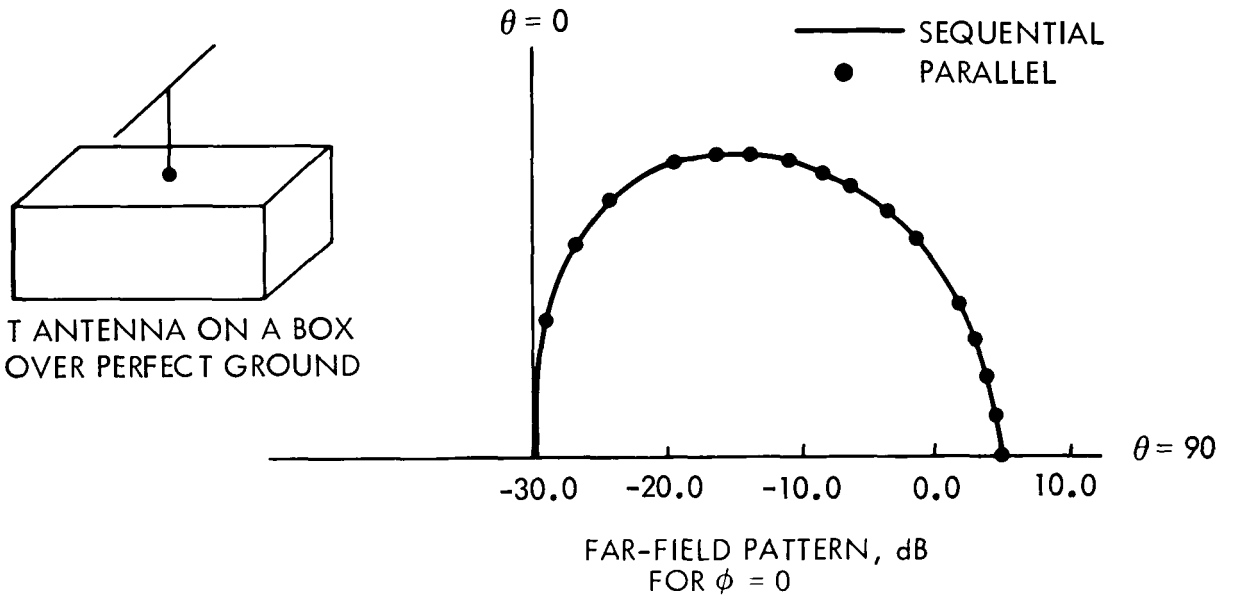


Figure 4.21. Radiation pattern of a T antenna on a box over perfect ground in the $\phi = 0$ plane

conducting wires are used, dots represent the case of thin aluminum wire (loading), and the Δ marks show the solutions for the case when thick wire is used with the extended thin wire kernel. All of these fields are calculated by the parallel code, and excellent agreement with the sequential code is verified for every case. It can be noted that the effect of the pedestal on the pattern is less significant when a fewer number of ribs are used (10 ribs here compared to 15 ribs in the example shown in Figure 4.19). Also, use of the extended thin wire, even with a lower number of ribs, models the pedestal much more realistically than the thin wire kernel modeling of Section IV.D.1. This more realistic model is evident from the strong interaction of the fields with the pedestal shown by Δ marks in Figure 4.22.

E. PERFORMANCE

To evaluate the performance of the NEC parallel program, one measure that can be used is the CPU time taken to run this code on the Mark III Hypercube compared with the time to run it on an VAX 11/750. Several parameters, such as the time taken to fill or factor the interaction matrix, and the speedup factor, are analyzed.

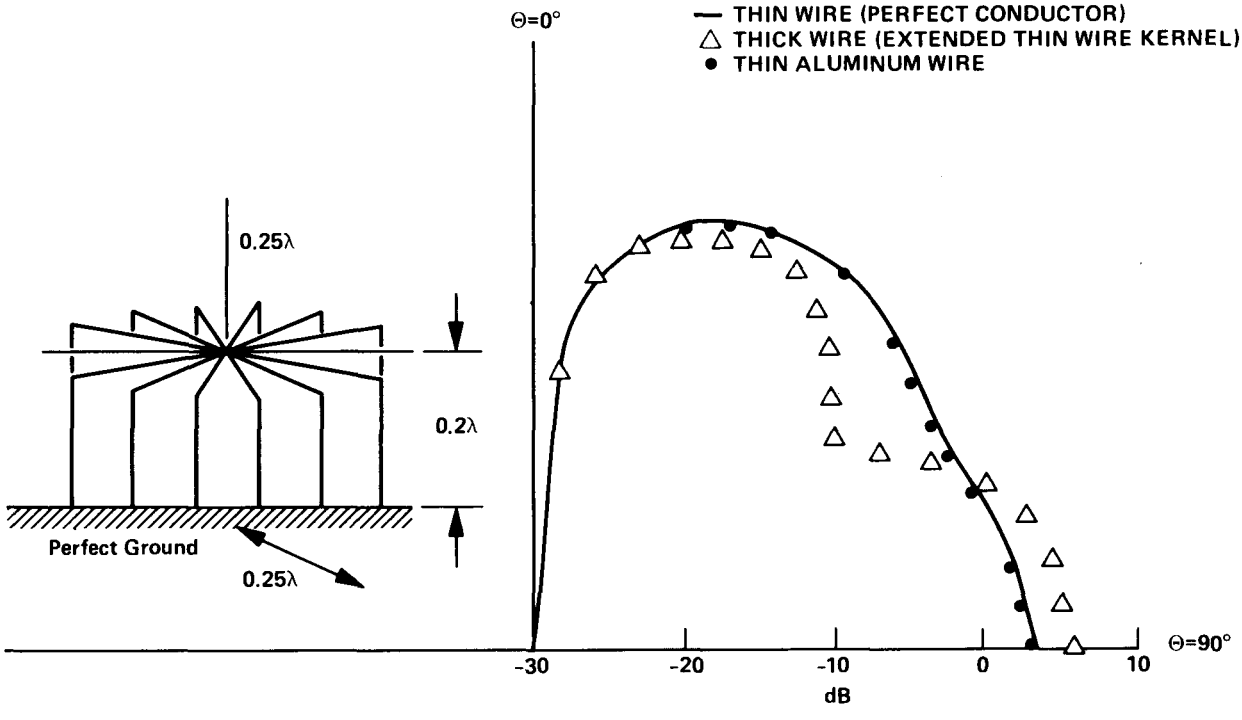


Figure 4.22. Radiation pattern of a monopole on a pedestal over perfect ground in the $\phi = 0$ plane

1. Timing

To analyze the timing performance of the parallel NEC program, two examples are considered:

a. Scattering by a Sphere. The sphere of Section IV.D.2 is modeled by 120 surface patches which do not use the symmetry option. The interaction matrix for this problem is of size 240 x 240. The times to fill and factor this interaction matrix, as well as the speedup factors, are shown in the tables of Figures 4.23 and 4.24, respectively,

DIMENSION OF CUBE	NUMBER OF NODES	FILL TIME, sec	SPEEDUP FACTOR
0	1	39.5	1
1	2	19.8	2.0
2	4	9.9	4.0
3	8	5.0	7.9
4	16	2.5	15.8
5	32	1.3	30.4

Figure 4.23. Time and speedup factor for filling the interaction matrix of a sphere modeled by 120 surface patches versus the dimension of the hypercube. The matrix size is 240 x 240.

as a function of the number of nodes used in the hypercube. In this case, as with the other timing runs in this section, unless otherwise noted, the fill is performed using the Source Loop Parallel Code (SLPC) and the Gaussian elimination technique for factoring.

The times taken to do the fill and the factor on one processor are 39.5 and 836.4 sec., respectively, while the same times are 25.4 sec. and 895.2 sec. for the VAX 11/750 computer.

Generally the fill and the factor times total more than 90% of the overall time in the NEC sequential program. Therefore the time taken to process the input, to solve the factored matrix, and to compute the far or near fields is small when compared to the overall time.

Currently the largest problem which can fit into one hypercube node is a modeled object requiring a total of 300 equations. This means that the limit is 300 wire segments or

DIMENSION OF CUBE	NUMBER OF NODES	FACTOR TIME, sec	SPEEDUP FACTOR
0	1	836.4	1
1	2	423.3	2.0
2	4	216.3	3.9
3	8	113.0	7.4
4	16	61.3	13.6
5	32	36.5	22.9

Figure 4.24. Time and speedup factor for factoring the interaction matrix of a sphere modeled by 120 surface patches versus the dimension of the hypercube. The matrix size is 240 x 240.

150 patches unless the symmetry option is invoked. The largest problem that the VAX 11/750 can run contains 300 equations. The 32-node Mark III Hypercube can run cases in core which consist of up to 2400 equations. The 128-node hypercube will be able to double this number to 4800 equations. With the addition of the disk drives to the Mark III Hypercube, out-of-core solutions can now be considered. With the 4 existing disk drives the maximum size problem which will be possible on the 32-node hypercube increases to about 10,000 equations.

b. Monopole on a Pedestal. In this section, the monopole on a pedestal of the type described in Section IV.D.1 is modeled by 20 ribs and a total of 290 segments. Therefore, the interaction matrix is of the size 290 x 290. The fill and factor times versus the number of nodes in use are shown in the tables of Figures 4.25 and 4.26, respectively.

DIMENSION OF CUBE	NUMBER OF NODES	FILL TIME, sec		SPEEDUP FACTOR	
		SOURCE-LOOP SEQUENTIAL	SOURCE-LOOP PARALLEL	SOURCE-LOOP SEQUENTIAL	SOURCE-LOOP PARALLEL
0	1	1725.0	1725.0	1	1
1	2	876.0	915.3	2.0	1.9
2	4	445.5	506.6	3.9	3.4
3	8	230.3	292.6	7.5	5.9
4	16	122.6	158.7	14.1	10.9
5	32	68.8	81.5	25.1	21.2

Figure 4.25. Time and speedup factor for filling the interaction matrix of the monopole on a pedestal versus the dimension of the hypercube. Results from source-loop parallel and source-loop sequential codes are shown. The matrix size is 290 x 290.

DIMENSION OF CUBE	NUMBER OF NODES	FACTOR TIME, sec	SPEEDUP FACTOR
0	1	1474	1
1	2	750	2.0
2	4	388	3.8
3	8	205	7.2
4	16	113	13.0
5	32	69	21.4

Figure 4.26. Factor time and speedup factor for the interaction matrix of the monopole on a pedestal versus the dimension of the cube. The matrix size is 290 x 290.

The speedup factors for filling and factoring the interaction matrix are plotted in Figure 4.27. The speedup factor is defined as:

$$\text{Speedup Factor} = \frac{\text{Time to run program on one node}}{\text{Time to run program on N nodes}}$$

For this problem, the top speedup factor for 32 nodes is 21.2 for filling and 21.4 for factoring.

For the fill time, the two parallel codes discussed in Section IV.C.2 are used. It is evident that for this problem, the source-loop sequential code is faster than the source-loop parallel code for any number of nodes used. For this example, processing of the information in the source loop takes less time than it takes to communicate that information to other nodes. The fill and factor times for the VAX 11/750 are 1808 sec. and 1771 sec., respectively. The fill time is very close to the time on one node because the parallel program for wires on one node is almost identical to the VAX sequential code. Additional data for timing and speedup factors for the 32-node Mark III Hypercube compared with those for the VAX 11/750 are shown in Tables 4.1 and 4.2.

The speedup factors for filling and factoring the interaction matrix versus the number of nodes used in the hypercube are shown in Figure 4.28. These curves are almost linear; however, the SLSC shows a better speedup factor across the range for the number of nodes than does the SLPC version.

2. Fixed Problem

When analyzing how a particular algorithm scales for different hypercube configurations, we need to keep the size of the problem fixed within a node. If the total problem size remains the same, the amount of work assigned to each processor decreases as the number of nodes in use increases. Consequently, as the number of nodes increases, the ratio of computation to communication decreases and the problem runs less efficiently. In other words, to run efficiently, it is important to keep the amount of computation high in each node.

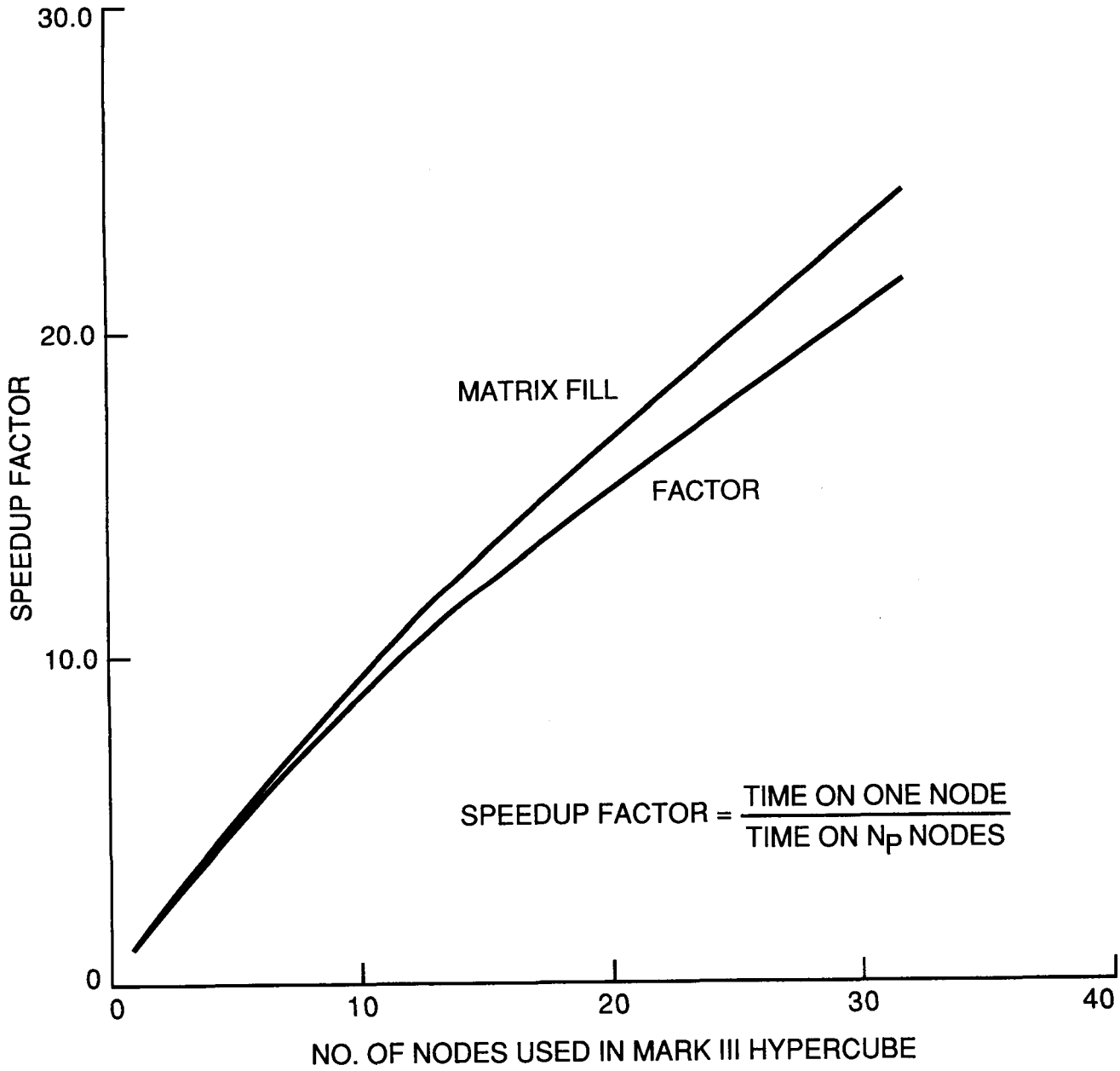


Figure 4.27. Speedup factor for filling (using SLSC) and factoring (using Gaussian elimination) the interaction matrix for scattering by a monopole on a pedestal versus the number of nodes used. The matrix size is 290 x 290.

Table 4.1. Timings for the sphere scatterer

Number of Patches	VAX, sec	Fill 32-Node, sec	Increase	VAX, sec	Factor 32-Node, sec	Increase	VAX, min	Total 32-Node, min	Increase
80	10.6	0.6	17.7	260.5	12.7	20.5	4.5	0.2	20.4
120	25.4	1.4	18.1	895.2	36.5	24.5	15.3	0.6	24.3
168	52.5	2.6	20.1	2661.0	91.7	29.0	45.2	1.6	28.8
224	95.1	4.5	21.1	6236.0	209.7	29.2	105.5	3.8	29.6
288	–	7.5	–	–	429.1	–	–	7.3	–
360	–	11.9	–	–	798.6	–	–	13.5	–
440	–	17.7	–	–	1431.6	–	–	24.2	–
528	–	25.6	–	–	2486.6	–	–	41.9	–
624	–	35.7	–	–	4056.2	–	–	68.2	–

Table 4.2. Timings for the monopole on a pedestal problem

Number of Wires	VAX, sec	Fill 32-Node, sec	Increase	VAX, sec	Factor 32-Node, sec	Increase	VAX, min	Total 32-Node, min	Increase
210	943.4	32.8	28.8	586.2	30.4	19.3	25.5	1.1	25.8
230	1125.5	39.9	28.2	817.6	38.0	21.5	32.4	1.3	24.9
250	1307.1	44.7	29.2	1078.9	48.7	22.2	39.8	1.6	25.5
270	1541.8	51.2	30.1	1423.6	56.7	25.1	49.4	1.8	27.5
290	1807.6	66.5	27.2	1771.1	69.1	25.6	59.0	2.3	26.4

To demonstrate how the FILL algorithm of the parallel NEC scaled with the size of the hypercube, the input data size is determined so that for each run all nodes receives 5000 elements, that is

$$\frac{n^2}{\text{No. of Nodes}} = 5000$$

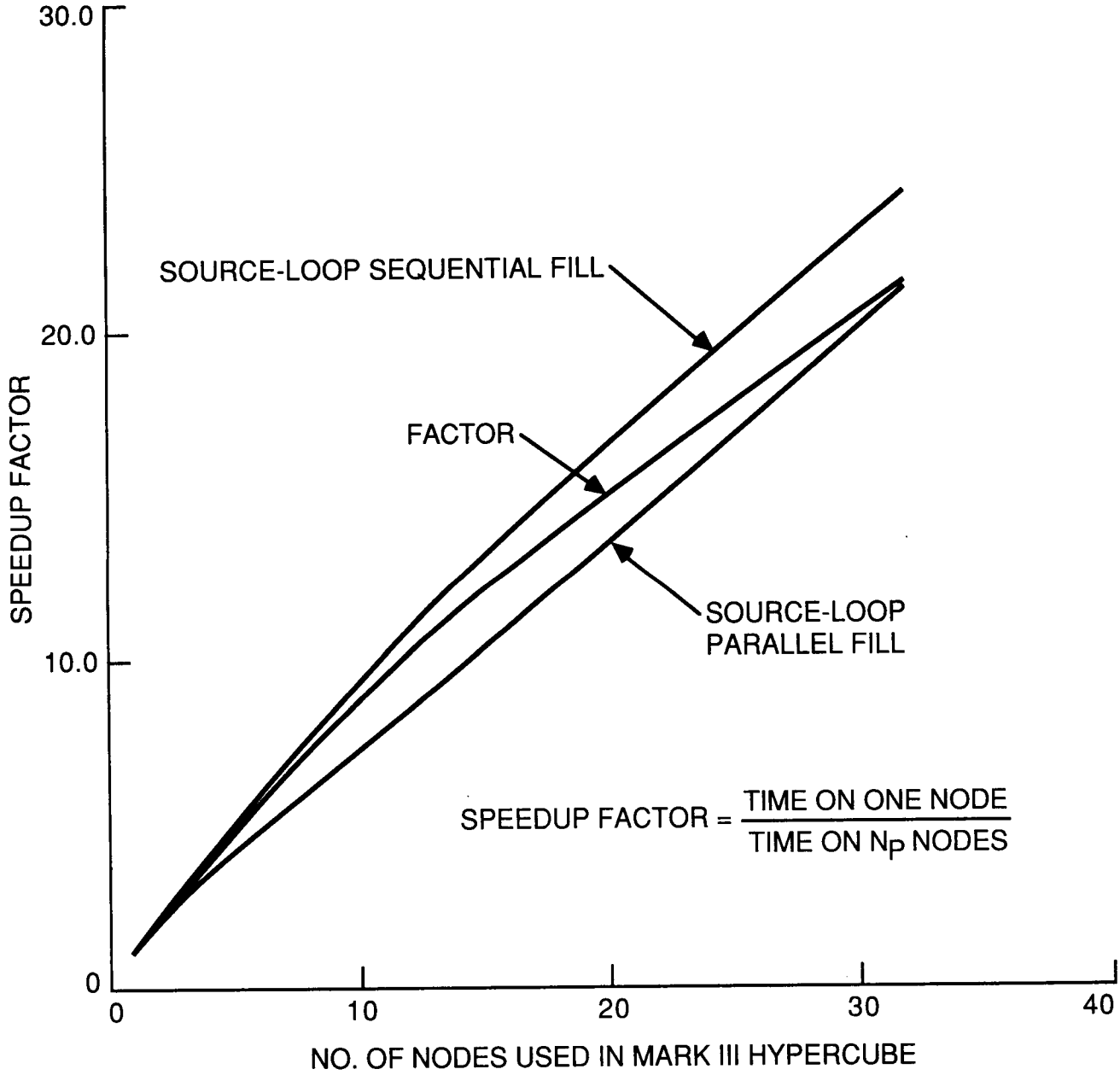


Figure 4.28. Speedup factor for filling and factoring the interaction matrix of the monopole on a pedestal versus the number of nodes used. Results from the source-loop parallel and source-loop sequential codes are shown for the matrix fill. The matrix size is 290 x 290.

As the number of elements in a row increases, the number of rows for which a node is responsible decreases. Figure 4.29 shows the results for the runs on the hypercube beginning with a dimension of 0 (1 node) and proceeding up to a dimension of 5 (32 nodes). As the number of nodes increases, the parallel algorithm demonstrates excellent scalability. The slight increase in time required for the larger size hypercubes would be expected. The number of elements per node is the product of N (the number of elements in a row of the interaction matrix) and the number of rows in that node. During the fill there is an initial computation which is performed N times. As the number of rows per node decreases, the size of N increases and, as such, affects slightly the overall time.

It is more difficult to analyze how the factor algorithm of the parallel NEC code scales with the size of the hypercube. One could construct a fixed case data set based on the fact that factorization algorithms are of order n^3. However, as the number of elements in the overall matrix increases for larger hypercubes, so does the number of transformation or elimination steps required to perform the factorization. Each step is preceded by communication of the pivotal row or column data. The results from different sizes of hypercubes would thus be a comparison of problems with considerably different computation and communication requirements.

3. Analysis of the Performance of Fill Algorithms

In using the two parallel codes, it is observed that for the 290-segments monopole on a pedestal, the source-loop sequential code (SLSC) consistently achieved lower fill times than the source-loop parallel code (SLPC). Even though better times are obtained by SLSC for most of the examples that we have run throughout this study, there are many cases for which SLPC yields lower fill times.

To understand this result better, we should study the source loop in more detail. As we discussed in Section IV.C.2, the source segment information is computed inside a DO-loop with index variable j, where j goes from 1 to N (the total number of segments). For problems where the structure is modeled by wires, the time to process this information can be substantial and therefore the SLPC was developed to communicate the data, instead of computing it sequentially in every node, as in the SLSC. For wire problems, each wire segment can be connected to several other segments as shown in Figure 4.30. In general, each segment can be connected to nseg number of other segments. Since the basis function

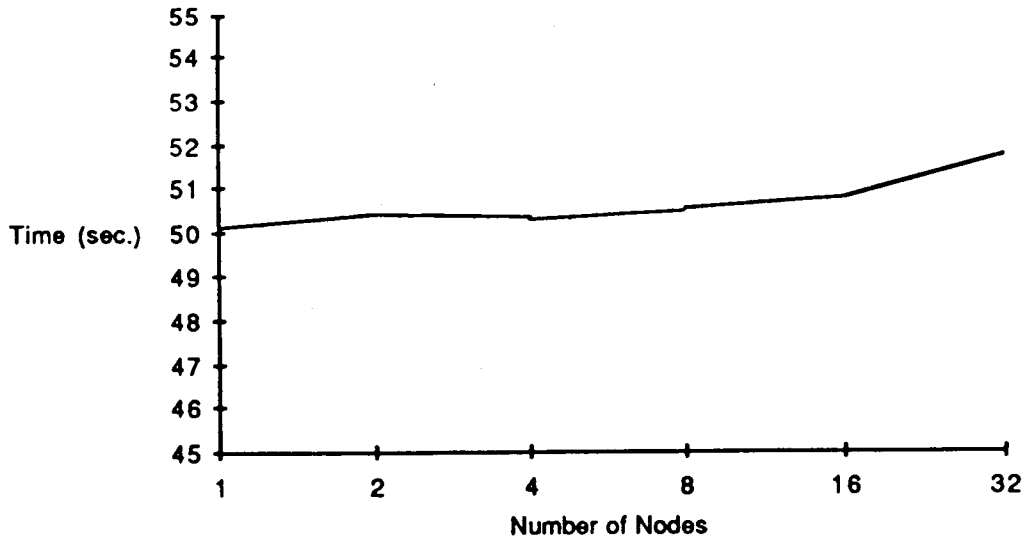


Figure 4.29. Scaling of the performance of the parallel fill algorithm where the problem size per node remains fixed at 5000 elements

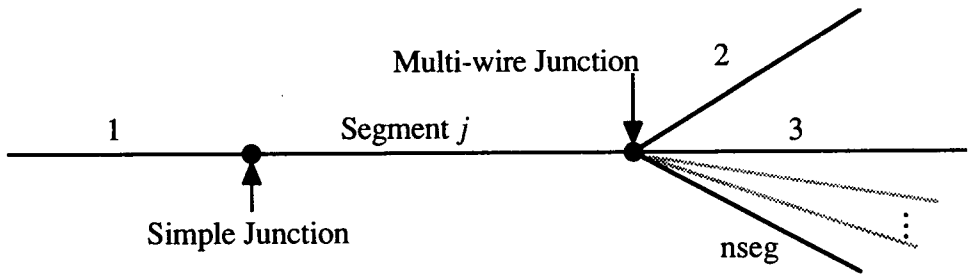


Figure 4.30. General wire segment connection

on each segment is extended over the adjacent segments, the number of local boundary equations to be solved for eliminating two out of three segments increases. In most problems, wire junctions have only 2 wires connected to them ("simple" junctions) and therefore source-loop information can be processed very fast without any need to use communication between nodes--SLSC should then be used. However, if a problem is modeled with multi-wire junctions, one has to look at the ratio of the number of these junctions to the number of simple junctions as well as the number of nodes used in the hypercube to select SLSC or SLPC.

For example, look at the monopole on the pedestal problem, which is modeled with a differing number of wire segments but always 20 ribs on the pedestal. Only one multi-wire junction with 21 segments is always present, and by changing the number of segments, only the number of simple junctions changes. These models were run on a 32-node hypercube, for maximum communication effect, and fill times obtained from SLSC and SLPC are plotted versus the total number of segments in the model, as shown in Figure 4.31. It can be seen that for a number of segments less than 90, the SLPC is faster. This speed is due to a relatively small number of simple junctions. As the number of segments is increased above 90, the number of simple junctions is increased and since, for these junctions, computing the source information is faster than communicating it, the SLSC yields better fill times.

F. FUTURE PLANS

There are three areas of emphasis in our current work on the method of moments code. The first is to implement an iterative Numerical Green's Function which has been described in Section IV.C.5. The ability to stepwise build and refine the design of an object by using previous calculations has now been made possible by the development of the Concurrent Input/Output system. Factored interaction matrices are archived in a distributed fashion on disk drives attached to the nodes. By having multiple disk drives, even large matrices (hundreds of megabytes in size) can be transferred to and from disk during the iterative design phase.

A second area of effort which has been enabled by the addition of disk drives to the hypercube nodes is the incorporation of out-of-core solutions. The maximum problem size which can be computed on the 32-node hypercube will expand from 2400 equations to about 10,000 equations. With the ongoing development of the hypercube operating system, virtual memory capabilities will eventually automate out-of-core solutions.

The third development which is under way is the parallel implementation of the output analysis routines. Up to this time these routines have run sequentially on the host computer, Counterpoint Control Processor, or on the Electromagnetic Interactive Analysis Workstation, a Sun 3/160. Because the time to perform the matrix computations has been dramatically reduced by the hypercube implementation, these sequential functions no longer

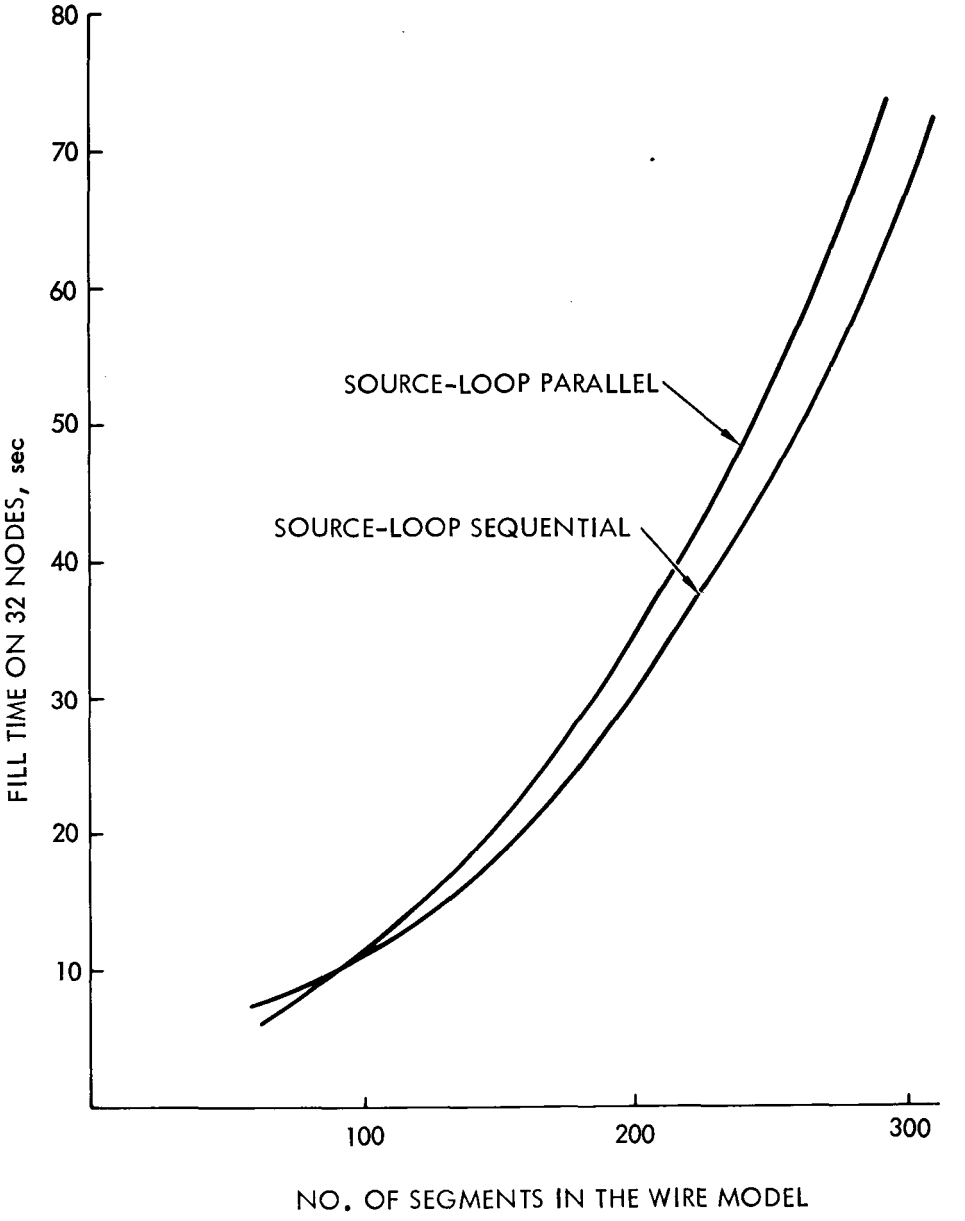


Figure 4.31. Time for the interaction matrix fill for a monopole on a pedestal versus the number of segments in the model for the source-loop parallel and source-loop sequential codes

take 10% of the overall time as they once did in the sequential code. Extrapolating from a law put forth by Amdahl, it is essential to put into parallel all aspects of the computation so that the once insignificant sequential computations do not become the dominant time component.

REFERENCES

[4-1] G. J. Burke and A. J. Poggio, "Numerical Electromagnetics Code (NEC) - Method of Moments," Lawrence Livermore National Laboratory, Livermore, CA, 1981.

[4-2] G. J. Bierman, Factorization Methods for Discrete Sequential Estimation--Mathematics in Science and Engineering, Volume 128, Academic Press, New York City, NY, 1977.

[4-3] E. Chu and A. George, "Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor," Parallel Computing, Proceedings of the International Conference on Vector and Parallel Computing--Issues in Applied Research and Development, June 1986, Loen, Norway, Nos. 1 and 2.

SECTION V

FINITE ELEMENT ANALYSIS

A. DESCRIPTION OF THE METHOD

1. Application of Finite Elements to Electromagnetic Scattering

The finite element method is used for the approximate solution of partial differential equations that arise in many engineering and scientific contexts [5-1]. As shown schematically in Figure 5.1, the method employs a spatial discretization of a continuous domain. In general, the domain is represented by a mesh of *nodal points*, and the polygonal (or polyhedral) regions delimited by this gridding are the *elements*, from which the finite element method derives its name. One of the principal strengths of the method is the ease and generality with which irregularly shaped domains can be treated, since elements of various shapes and sizes may be employed.

We shall assume that the original mathematical statement of the problem takes the form of an elliptic partial differential equation. In the context of electromagnetic (EM) scattering, this differential equation is the Helmholtz equation describing the incident and scattered fields for a particular wave number, k. As a concrete example, we shall consider the two-dimensional (2-d) scattering of an incident transverse electric (TE) polarized wave by an arbitrary 2-d body. In this case, the out-of-plane magnetic field H_z satisfies the Helmholtz equation:

$$\nabla\left(\frac{1}{\varepsilon}\nabla H_z\right) + k^2 H_z = 0 \tag{5.1}$$

where ε is the relative permittivity and k is the wave number. In practice, this open region problem is solved within a finite domain through the introduction of an artificial far-field boundary condition. In the present work, we have adopted the approach of Bayliss and Turkel [5-2]. A cylindrical artificial boundary is parameterized by radius ρ and angular coordinate ϕ. The boundary condition on scattered fields then becomes:

$$\frac{\partial H_z}{\partial \rho} = A(\rho) H_z + B(\rho) \frac{\partial^2 H_z}{\partial \phi^2} \tag{5.2}$$

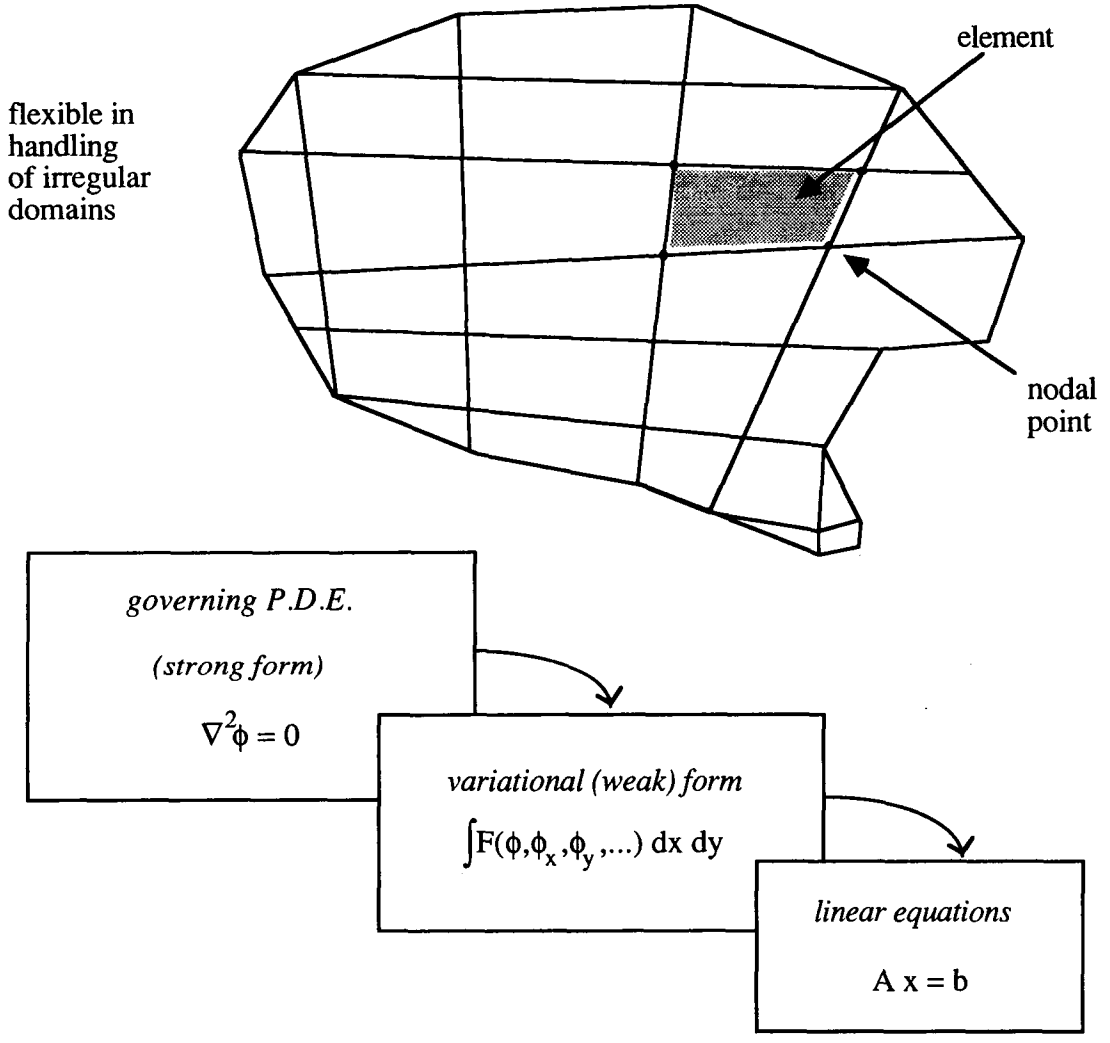


Figure 5.1. The finite element method employs a discretized spatial domain for the approximate solution of boundary value problems. Unknown field quantities are defined at nodal points, and interaction matrix elements are derived by integration over finite elements. The discretization leads to a system of linear algebraic equations.

where A and B are ρ-dependent operators. We turn now to the finite element approximation of these original equations.

The governing differential equation or so-called *strong form* can be shown to be equivalent to an integral variational *weak form* statement of the problem. In this particular case, the weak form is an integral equation of the form:

$$\int\int_{\Gamma}\left(\frac{1}{\varepsilon}\nabla T \bullet \nabla H_z^s - k^2 T H_z^s\right) dx\, dy - \oint_{\partial\Gamma} \frac{1}{\varepsilon} T\left(A H_z^s + B\frac{\partial^2 H_z^s}{\partial \phi^2}\right) dl = F \qquad (5.3)$$

where T is an arbitrary test function that satisfies certain continuity conditions. Equation (5.3) contains the unknown scattered H field in the left hand side, while the excitation term in the right hand side depends on the incident field through:

$$F = \int\int_{\Gamma}\left(T\nabla \bullet \left(\frac{1}{\varepsilon}\nabla H_z^{inc}\right) + k^2 T H_z^{inc}\right) dx\, dy \qquad (5.4)$$

Next, this weak form is applied to the discretized finite element domain through the introduction of a set of approximate nodal basis functions. This discretization of the weak form results in a discrete set of unknown coefficients that are related by a system of linear algebraic equations. Thus, the finite element method involves the construction and solution of a matrix system whose rank is equal to the total number of unknown nodal degrees of freedom.

The matrix which results from this finite element approximation is, in general, sparse, with an irregular clustering of non-zero elements near the diagonal. The particular structure and non-zero column profile of the *stiffness matrix* is dictated by the spatial connectivity of the problem domain. The matrix contains non-zero entries connecting only those degrees of freedom which share a common finite element in the domain discretization. This spatial locality results from the element-wise assembly of the stiffness matrix. Element-level matrix contributions are obtained by integration of basis functions over the volume of individual elements, which are in turn additively assembled into the global matrix.

The central computational task from a standpoint of cost is the solution of this matrix system. While the overall design and implementation of the supporting finite element software superstructure is also important, the emphasis of the work described here is on the equation solution task. In this and the following sections, methods for the efficient parallel solution of large finite element linear systems are described.

2. Finite Elements in the Context of Parallel Processing

As stated above, the principal computational cost associated with the finite element method is the solution of a large matrix system. The cost-effective implementation of this task (on either a sequential or parallel computer) requires a recognition of the special structure and properties of the finite element matrix, in contrast with the case of a general full matrix. One possible approach to the parallelization of a sparse matrix solver is to decompose (or break up) the task by matrix rows and columns. While this is a viable approach for some applications [5-3], it has disadvantages for finite element matrices.

One of these disadvantages is the inherent irregularity of the non-zero matrix element profile, which arises from the ability of the finite element method to treat geometrically complex grids. This irregularity complicates the problems associated with efficiently programming a row/column decomposition and maintaining an even work load balance among the parallel processors. A perhaps more obvious disadvantage is the relatively large amount of interprocessor data communication required to perform operations on rows and columns that reside in different processors. (While this problem is most apparent in a distributed memory concurrent architecture, it can also manifest itself in shared memory machines in the form of memory access conflicts.)

In the present research, we utilize a domain decomposition approach, which divides up the problem in the physical domain space, rather than in the abstract "matrix world." The meaning of domain decomposition in the current context is illustrated in Figure 5.2. Contiguous spatial subdomains of finite elements are assigned to the distinct processor nodes of a multiprocessor computer. We need to be somewhat more specific, however, about what is meant by "assignment." Our machine model for decompositions of this kind is based upon large-node, distributed memory architectures like that of most hypercube multicomputers. In this model, all the information relevant to nodal degrees of freedom or

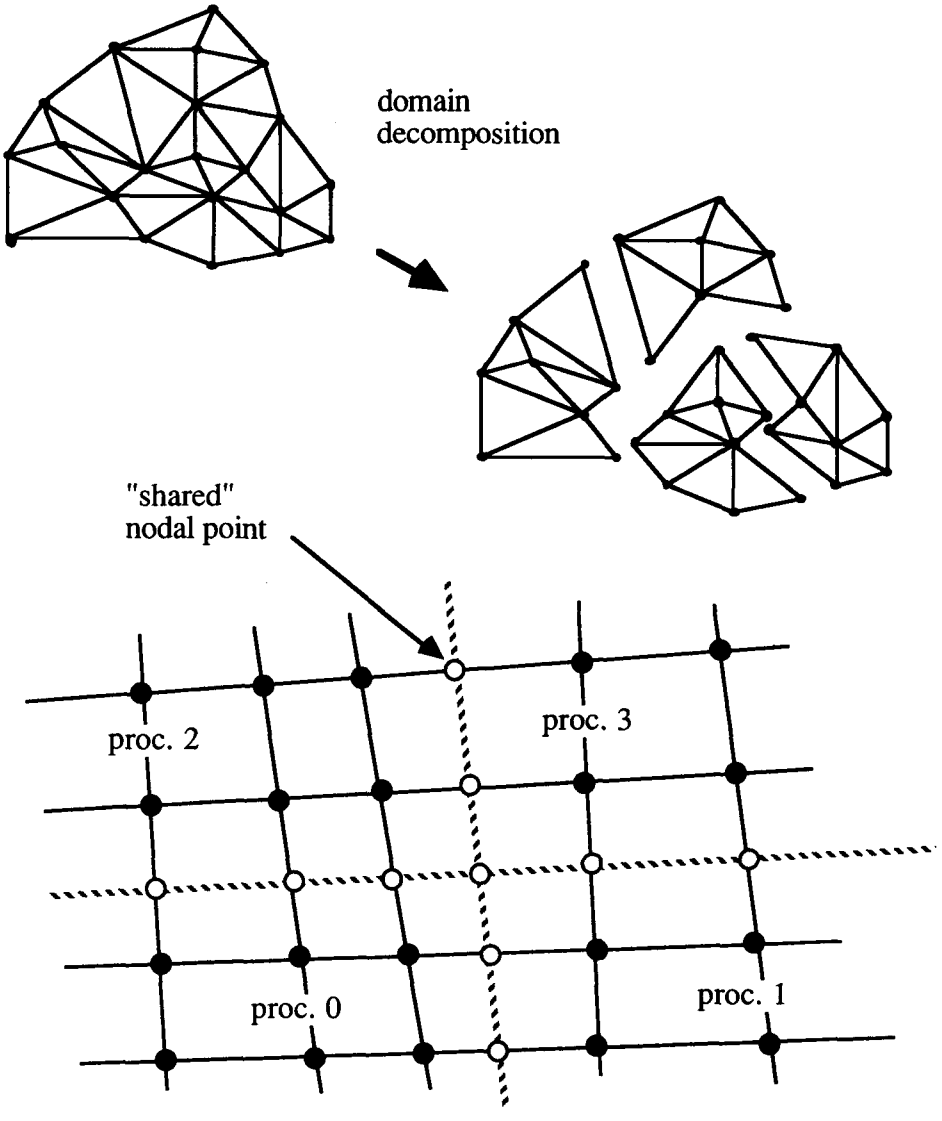


Figure 5.2. Domain decomposition divides the finite element grid into subdomains assigned to different processors. Elements are the exclusive responsibility of a single processor, while some nodal points (open) are shared between processors.

matrix elements resides in the local memory of the processor corresponding to the given subdomain. A given processor has direct access only to its "own" information and can be thought of as solving a smaller version of the original full problem, subject to some special boundary conditions. Interprocessor communication is invoked in order to exchange this "boundary" information and obtain a global solution.

Examining the finite element decomposition in more detail, we see that *elements* become the exclusive responsibility of exactly one processor. While *nodal point* information is also distributed among processors, it is seen that some nodal points fall on boundaries and are in some sense shared by two or more processors. In such cases, it is necessary to devise a bookkeeping scheme to ensure consistent treatment of these degrees of freedom. A detailed description of the programming used to accomplish this management is beyond the scope of the present report, but has been described elsewhere [5-4]. For the purposes of the present discussion, it is sufficient to say that the data structures and constructs needed to keep track of processor assignments fit naturally within the same structures used in "traditional" finite element applications to manage different element types and material property variations. It is only necessary in this context to assume that any pair of processors in the concurrent computer can exchange "packets" of communicated data via a kind of "mail" system. In the case of the hypercube computer operating in the "crystalline" or polled communication environment, this functionality is achieved through routed message passing (the so-called Crystal Router [5-3]) and can be regarded as a transparent "operating system" task.

This domain decomposition approach to parallel finite elements considerably lessens the interprocessor communication load over a row/column decomposition. In this case, communication becomes essentially an edge effect, the relative magnitude of which diminishes with increasing problem size as the domain perimeter-to-area ratio decreases. Because of the spatial locality of the finite element matrix elements, only those elements spatially adjacent to a processor subdomain boundary need be involved in communication activity.

Having discussed the rationale for domain decomposition, we turn next to the particular linear algebraic methods used to solve the matrix system. One common and well-understood approach to the solution of large linear systems is *direct elimination.* Variants of Gaussian elimination which exploit the limited non-zero column height of the sparse finite element matrix are widely used. Useful because of their comparative robustness and cost-predictability, direct elimination methods can be prohibitively expensive for systems of large rank and bandwidth, such as arise from large three-dimensional domains. It is possible to adapt such methods to a domain-decomposed parallel setting by treating the subdomains as "substructures" which interact through their mutually shared boundary

nodes. Gaussian elimination may be employed concurrently within each processor to eliminate the interior (non-shared) degrees of freedom. Interprocessor communication of the remaining (potentially large) boundary matrix would then ensue, allowing for the subsequent elimination of the remaining shared nodal values.

An alternative approach to linear equation solving which also employs domain decomposition is the use of iterative methods. Iterative solvers (e.g., the conjugate gradient method) proceed by iterative improvement through a succession of approximations toward an ultimate solution. A good deal of previous work has gone into study of such methods for parallel implementation because of two considerations. Firstly, the computational cost (per iteration) of such methods increases much less rapidly with increasing domain size than is the case for direct methods. Secondly, the iterative methods are most amenable to domain decomposition, requiring only minimal communication of boundary information between processors. For these reasons, the work described here has initially focused on iterative methods, although in some applications, direct or direct/iterative hybrid methods may prove useful.

Another issue which arises in the consideration of direct vs. iterative solvers is the ability to efficiently solve for multiple right-hand sides (excitations). This is an issue of some importance in EM scattering applications, in which it is often desirable to systematically examine a range of incident field parameters using the same matrix. Although direct factorization / backsubstitution methods have been historically used for the multiple excitation problem, it has been shown [5-5] that the conjugate gradient method can be formulated to efficiently solve for multiple right-hand sides as well. These questions are among those designated for examination in the near future.

Unfortunately, iterative methods are not particularly robust, in the sense that their convergence properties are highly problem-dependent and difficult to guarantee with any single preconditioner. For this reason, the performance of these methods independent of parallel processing issues should be ascertained prior to attempting their routine application to a particular class of problems. The following subsections take up the application of the method of conjugate gradients (and derivatives thereof) to the solution of linear systems of complex equations.

3. Solution by the Conjugate Gradient Method

As shown above, applying the finite element method to EM scattering situations results in the linear system of equations $Ax = b$. The coefficient matrix A is usually a sparse, large matrix within the finite element approximation, resulting in the need to use specialized methods of solution suited for this situation. Direct methods, such as Gaussian elimination, are most suitable and effective for relatively small-scale problems. As we have seen, however, as the size of the matrix increases, storage and computation costs become excessive, and it becomes necessary to consider iterative methods. The following is a detailed description of one such important method, the method of conjugate gradients.

Within the family of gradient methods, the process of solving a set of simultaneous equations is equivalent to that of finding the position of the minimum for an error function defined over an N-dimensional space (N being the order of the matrix). In each step of a gradient method, a trial set of values for the variables is used to generate a new set corresponding to a hopefully lower value of the error function. The natural error function is the square of the norm of the residual at each step, $r^{(k)} = b - Ax^{(k)}$. The best known gradient method is the steepest descent method, in which the search direction at the k^{th} step is chosen to be the direction of the maximum gradient of the error function at the point $x^{(k)}$. It has been established [5-6] that the steepest descent method suffers from the following drawback: the method will perform many small steps in going down a long, narrow valley, even if the valley is a perfect quadratic form.

In the method of conjugate gradients [5-6] , designed to overcome this difficulty, the direction vectors are chosen to be a set of vectors representing, as nearly as possible, the directions of steepest descent of the points $x^{(0)}$, $x^{(1)}$, ..., but with the overriding condition that they be mutually conjugate. Here the term conjugate means that the vectors are orthogonal with respect to the inner product weighted by matrix A. The basic algorithm, assuming a real, symmetric, and sign-definite matrix A, is given by the following set of equations:

0. Define for k = 0

Initial Guess: $\mathbf{x}^{(0)}$

Initial Residual: $\mathbf{r}^{(0)} = \mathbf{b} - A\,\mathbf{x}^{(0)}$

Search Direction: $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$

1. $$\alpha_k = \frac{\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)}}{\mathbf{p}^{(k)} \cdot A\,\mathbf{p}^{(k)}}$$

2. $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$$

3. $$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A\,\mathbf{p}^{(k)} \qquad (5.5)$$

4. $$\beta_k = \frac{\mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}}{\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)}}$$

5. $$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta_k \mathbf{p}^{(k)}$$

6. $k \rightarrow k+1$, go to step 1.

Because of the orthogonal relationship, the exact solution is obtained after exactly N steps for infinite machine accuracy; the method should not, strictly speaking, be classified as iterative. It cannot be assumed, however, that convergence will always be obtained after exactly N steps. In some cases convergence to an acceptable accuracy will be obtained after less than N steps, while in other cases rounding errors or intrinsic ill-conditioning of the problem at hand will affect the computation to such an extent that more than N steps will need to be performed. For these reasons the conjugate gradient method is treated and programmed as an iterative method. The various numerical approaches to large-size matrices occurring in EM scattering have been reviewed by Sarkar [5-7] and Van den Berg [5-8]. The use of the conjugate gradient method in parallel processing finite element applications has been demonstrated recently by Fox [5-3], Nour-Omid [5-10], and collaborators.

Another problem arises when the matrix A is not real or symmetric (the matrix is complex in our EM scattering problem). To reduce the problem to the required conditions,

Peterson and Mittra [5-9] describe a modified conjugate gradient algorithm, in which the original matrix A is multiplied by its adjoint, resulting in an algorithm in which the matrix A appears not linearly, as in the original formulation, but quadratically. This creates a serious problem for ill-conditioned matrices, for which the condition number is large: the condition number of the matrix A^TA is the square of that of the original matrix!

4. Extensions of the Conjugate Gradient Method

An important issue related to iterative methods is to understand and improve their rate of convergence. While the iterative approach permits us to tolerate loss of orthogonality, it does make the issue of rate of convergence critically important because it suggests a hopelessly slow rate of convergence for ill-conditioned problems. An important way around this difficulty is to precondition the matrix A. This refers to finding a nonsingular symmetric (or self-adjoint in the complex case) matrix C such that $A' = C^{-1}AC^{-1}$ has improved conditioning. We can then apply the conjugate gradient (with improved convergence) to the transformed system

$$A'x' = b' \tag{5-6}$$

where $x = C^{-1}x'$ and $b' = C^{-1}b$. With C^{-1} being the approximate inverse of the original matrix A, the coefficient matrix $C^{-1}A$ of the modified system $C^{-1}Ax = C^{-1}b$ is very close to the identity matrix, and hence will have eigenvalues much more closely clustered than those of the original matrix A. For problems in which the conditioning number of the original system is too high, this can result in significant reduction in the conditioning number of the problem and, subsequently, improvement in the convergence rate. One of the commonly used preconditioners is diagonal scaling, in which the matrix C consists of the square roots of the diagonal elements of the original matrix A. This causes the elements of the transformed A' to be of the order of unity, corresponding to a much smaller condition number and faster convergence of the conjugate gradient iteration scheme.

Another important situation occurs when the coefficient matrix A is not real, symmetric, and sign-definite at the same time. This is the case in our problem, in which many of the coefficients are complex. When applying preconditioned conjugate gradient methods to complex systems, there are several approaches that could be used. The variables could be split up into their real and imaginary components to generate a real

system of twice the size. It is preferable, however, to work with complex quantities directly. The preconditioned system obtained will consist of complex equations. The 'best' approach is to develop a version of conjugate gradients applicable to complex systems of equations. The method of bi-conjugate gradients was developed for non-symmetric, indefinite systems. The algorithm, developed by Jacobs [5-11], is given by

0. Define for k = 0

   Initial Guess: $\mathbf{x}^{(0)}$

   Initial Residual: $\mathbf{r}^{(0)} = \mathbf{b} - A\,\mathbf{x}^{(0)}$

   Initial Bi-residual: $\bar{\mathbf{r}}^{(0)} = \mathbf{r}^{(0)}$ *

   Search Direction: $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$

   Search Bi-direction: $\bar{\mathbf{p}}^{(0)} = \bar{\mathbf{r}}^{(0)}$

1. $$\alpha_k = \frac{\bar{\mathbf{r}}^{(k)} \cdot \mathbf{r}^{(k)}}{\bar{\mathbf{p}}^{(k)} \cdot A\,\mathbf{p}^{(k)}}$$

2. $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$$

3. $$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A\,\mathbf{p}^{(k)} \qquad (5.7)$$

   $$\bar{\mathbf{r}}^{(k+1)} = \bar{\mathbf{r}}^{(k)} - \alpha_k^{*} A^{T} \bar{\mathbf{p}}^{(k)}$$

4. $$\beta_k = \frac{\bar{\mathbf{r}}^{(k+1)} \cdot \mathbf{r}^{(k+1)}}{\bar{\mathbf{r}}^{(k)} \cdot \mathbf{r}^{(k)}}$$

5. $$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta_k \mathbf{p}^{(k)}$$

   $$\bar{\mathbf{p}}^{(k+1)} = \bar{\mathbf{r}}^{(k+1)} + \beta_k^{*} \bar{\mathbf{p}}^{(k)}$$

6. $k \rightarrow k+1$, go to step 1

where the superscript T denotes the transpose of the vector or matrix of complex conjugate elements, the inner product of two vectors is given by $x \cdot y = x^T y$, and * denotes the

complex conjugate. It should be noted that the update and direction coefficients are now complex coefficients, and that the dimensionality of the residual vectors and of the search directions is now effectively doubled, resulting in more degrees of freedom for solving the problem. This is compared and contrasted with Peterson and Mittra [5-9], in whose algorithm these coefficients are real, and the possibly ill-conditioned matrix A appears quadratically.

Other features of the bi-conjugate gradient method are: 1) When no loss occurs in the dielectric scatterer, the complex matrix A becomes real. The problem is still complex, however, because of complex terms in the right-hand-side excitation (the forcing term F in Eq. (5.4)). 2) The coefficient matrix A in our electromagnetic scattering problem is complex symmetric. In this case the bi-residuals and bi-directions are the complex conjugates of the original respective quantities. This symmetry property can be utilized to reduce the number of arithmetic operations per iteration, as it is not necessary to implement the full algorithm.

Finally, we note that the best results can be expected when one uses the preconditioned bi-conjugate gradient (PBCG) method, in which both approaches are put at work simultaneously. This will be demonstrated below.

B. INITIAL RESULTS FOR TWO-DIMENSIONAL SCATTERING

1. Description of the Problem

A simple 2-d test problem was selected for the initial investigation of the finite element solution methods outlined in the previous section. Since these initial calculations were aimed at evaluating the algorithms prior to their parallel implementation on the hypercube, the test problem is comparatively small and runs quickly on a conventional sequential processor. This employment of a scaled-down test grid provides a good test of various iterative methods relative to one another, since the basic finite element structure is embodied in the test grid. The absolute comparison of this example with direct elimination results is not as straightforward, however, because the cost advantages of iterative methods are generally realized only in larger problems.

The test case under consideration is illustrated in Figure 5.3. A dielectric circular cylinder of radius $\lambda/4$ is modeled by a mesh of triangular patch elements. A free space

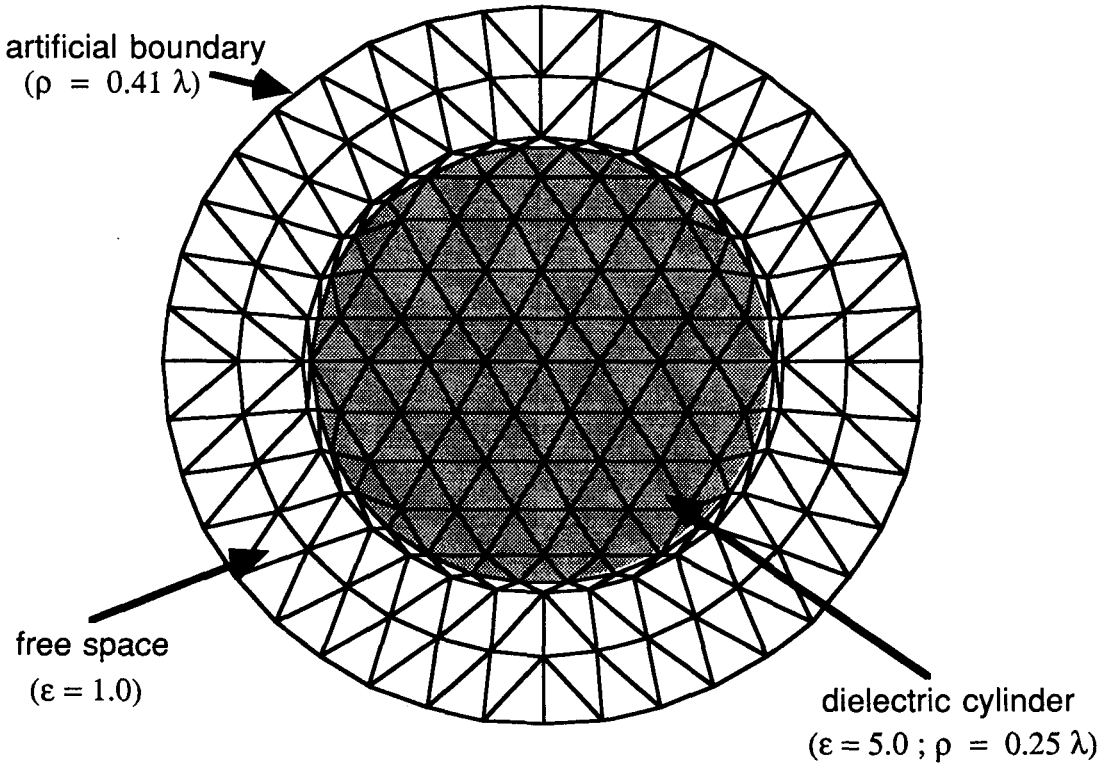


Figure 5.3. Finite element grid of the 2-d cylinder test problem domain. Grid is composed of 193 nodal points at which field quantities are evaluated and 344 triangular finite elements (or patches). The case calculated is scattering from a lossless dielectric cylinder, although more general geometries and scatterer properties are handled by the present program.

region with an artificial boundary radius of 0.41 λ surrounds the scatterer. While the finite element code is capable of as easily modeling more irregular geometries, this simple case provides an adequate technique evaluation benchmark.

The grid is composed of 193 nodal points comprising 344 elements. An incident TE-polarized plane wave of wavelength λ defines the excitation. In the particular case considered here, the permittivity of the dielectric is a real number (5.0), corresponding to a lossless scatterer. This case is considered representative of typical calculations, although it should be noted that different permittivity values somewhat alter the finite element matrix, having a noticeable effect in some cases on the rate of convergence of iterative solvers.

2. Solution Methods

The finite element calculation described here breaks down naturally into four tasks: 1) formation of the left-hand-side matrix elements, 2) formation of the right-hand-side excitation vector, 3) the actual solution of the linear system, and 4) calculation of the scattering cross-section from the resultant fields. Of these tasks, 1), 2), and 4) have been implemented essentially without any change from their usual formulations. The matrix is formed using geometric information about the grid and element permittivities, while the excitation is derived from the assumed incident field(s). In the present work, only the solution task 3) is novel, and the subsequent discussion centers on this task.

The following iterative schemes were employed in succession to the above-described problem: 1) simple conjugate gradient (CG) applied to the complex case [5-9], 2) preconditioned conjugate gradient with diagonal scaling (PCG), 3) the bi-conjugate gradient (BCG) method, and finally 4) preconditioned (diagonally scaled) bi-conjugate gradient (PBCG). The programming required to implement these solvers is quite straightforward, as outlined below.

In each of these conjugate gradient variants, the coefficient matrix A is involved strictly in multiplicative operations of the form Ax, where x is an N-vector. Because of this fact, it is sufficient to store the matrix in the form of separate (unassembled) element submatrices, and multiplications are accomplished by a simple linear combination of small element contributions. This strategy simplifies implementation of the conjugate gradient method in sequential as well as parallel applications. For any one of the four specific solvers listed above, it is necessary to define the N-vectors (containing residuals, search directions, etc.) used for the particular algorithm, and then program the corresponding sequence of scalar and matrix-vector products. In the case of the two diagonally preconditioned methods, two additional computational steps intervene. Prior to beginning the iterations, the matrix A and right-hand-side are rescaled appropriately by the diagonal elements of the original system. Following completion of the iterations, the resultant solution must be scaled back again to yield the solution originally sought.

In applying these conjugate gradient methods for a comparative benchmark test, it was necessary to define a uniform convergence criterion. This criterion was chosen as that number of iterations required to obtain a solution which differed negligibly from the

solution of the same system obtained by Gaussian elimination. In practice, this amounted to reducing the Euclidean norm of the initial residual vector by a factor of approximately 10^{-5}. The performance results obtained in each of these cases are discussed in the next subsection.

3. Comparative Performance of Solution Methods

As mentioned above, the test case used in this report is that of TE-wave scattering from a dielectric cylinder. The finite element code for this situation, generalized to the case of a 2-d body with an arbitrary cross section, and followed by solving the system of resulting linear equations by Gaussian elimination, was kindly provided to us by Dr. A. F. Peterson [5-14]. Our initial effort has centered on replacing the Gaussian elimination solver by a variety of conjugate gradient solvers, anticipating that some member(s) of this family will be the natural candidate for solving larger-scale problems by parallel processing.

The convergence rates for the four different conjugate gradients used so far are illustrated in Figure 5.4, in which the relative residual norm is plotted as a function of the number of iterations. It is seen that the CG algorithm presented by Peterson and Mittra [5-9], and based on extending the basic formulation for real, symmetric, and sign-definite matrices, converges too slowly in comparison with the theoretical limit of no more than about 200 iterations. The reason for this is that we deal with a complex, possibly poorly conditioned, matrix. The next step was to apply diagonal preconditioning [5-12], which improved the convergence dramatically, bringing the rate down to the upper theoretical limit for the CG algorithm.

Further substantial improvement resulted from realizing that for a complex matrix we need a reformulated algorithm, which is provided by the bi-conjugate gradient method. We used the formulation provided by Jacobs [5-11] and recently used by Smith, Peterson, and Mittra [5-17]. The last two curves in Figure 5-4 show that the use of bi-conjugate gradients provides convergence rates of the quality of those reported by Peterson and Mittra [5-9] in their case studies involving smaller fully populated matrices resulting from the use of the method of moments. For the two bi-conjugate gradient versions (BCG and PBCG) we used both the full Jacobs algorithm [5-11] for a general complex matrix, and a

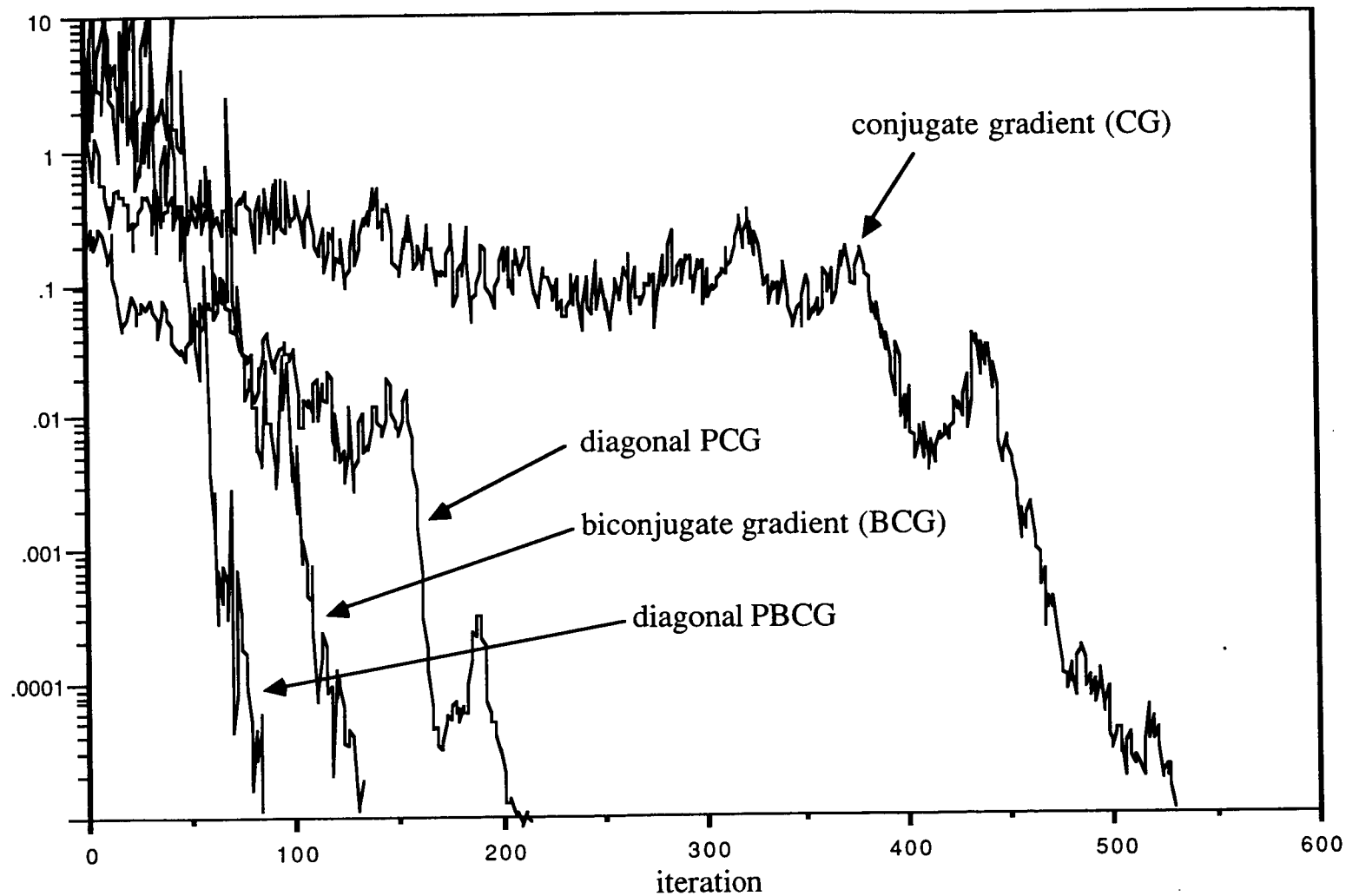


Figure 5.4. Convergence histories for conjugate gradient methods applied to the finite element system arising from the grid of Figure 5.3. The Euclidean norm of the residual vector is plotted as a function of the iteration number for the four CG variants investigated.

simplified algorithm for a complex symmetric matrix, resulting in the cutting by half of the number of matrix-vector multiplications per iteration and, respectively, the speeding up of the calculation by nearly a factor of two. The question is still open whether the use of non-diagonal preconditioning might further improve the convergence rate of bi-conjugate gradients. In this context we note that preconditioned conjugate gradients have been recently applied [5-15] to problems of frequency selective surfaces and scattering from dielectric cylinders (more precisely, a TM-polarized wave scattered from a square cylinder). The nature of the improvement of the convergence rate due to preconditioning is similar to ours, but further study is required to compare the two studies and explore the possibilities of non-diagonal preconditioning.

Compared with direct Gaussian elimination, the diagonal PBCG method still takes three times longer to converge than direct Gaussian elimination does. However, it is expected that with more complicated situations, in general, and with 3-d scattering, in particular, the additional effort and computation costs associated with iterative methods will increase much more gradually than indicated by experience with direct methods. This has been our major motivation for exploring the different conjugate gradient variants.

C. PLANS FOR FUTURE WORK

1. Parallel Program Development

The next immediate goal of the finite element analysis task is the implementation of the above-described results on the hypercube computer. The principal areas of effort in this near-term work involve construction of the parallel finite element code proper, and the development of input decomposition tools that will serve as the "front end" to the finite element code. The issues associated with these developments are discussed below.

Much of the "superstructure" finite element programming needed for the hypercube code is directly transferable from already completed work in finite element stress/strain analysis [5-4]. Only those parts of the programming uniquely related to the electromagnetics application will require extensive rewriting. Routed crystalline communications will be used to implement the various conjugate gradient methods in parallel. The parallel efficiency of these iterative approaches with domain decomposition has been well documented already [5-3], and is not considered a central issue at this stage.

It is to be expected that finite element calculations with concurrent efficiencies exceeding 90% for reasonable problem sizes will be realized. Of more direct interest are questions associated with accelerating the iterative solution procedure, and with facilitating the problem decomposition process.

Augmentation of the conjugate gradient methods described above by preconditioning will form an important part of the next year's work. A variety of non-diagonal preconditioning approaches [5-13] are currently being examined for parallel implementation. In addition to work on preconditioning, work will be devoted to departures from the basic conjugate gradient formulation. One such generalization is the modification to allow solution of multiple excitations efficiently, as described earlier in Section V.A. Another area of considerable interest is the departure from a purely iterative scheme for hybrid direct / iterative methods. Previous work in hypercube finite element stress analysis suggests that combining Gaussian elimination on subdomain interiors with conjugate gradient iteration on interprocessor boundaries can accrue many of the benefits of both direct and iterative solvers. Extension of these hybrid techniques to the complex electromagnetics setting represents an important priority following development and demonstration of the basic parallel iterative code.

Input decomposition represents the other important area for research and development in the next year. Among the desiderata in a concurrent finite element analysis package is a near-transparency at the user level to the fact that the problem is decomposed. This implies a certain amount of automation (or at least computer aid) in the decomposition process. While "carving up" the domain for decomposition is a conceptually fairly simple process, doing so rapidly and optimally may be difficult in practice. Our approach in the next year will be to integrate the domain decomposition process with the grid generation task. Processor subdomain assignment information is easily and naturally couched within the already existing conventions for finite element mesh definition. The development of basic graphical tools for the on-screen interactive definition of decomposition subdomains will represent a significant advance in this regard, eliminating the necessity to decompose conventional finite element input data "by hand."

2. Proposed Test Problems

As just discussed, the main thrust in the immediate future will be to complete the test case currently under way. However, a number of additional test problems come to mind and will be addressed as this stage of the investigation nears completion. A particularly interesting problem is that of scattering from a conducting cylinder with a dielectric coating. This problem has been treated analytically in the last two decades [5-18], and some of these analytic results will need to be recoded, so as to provide numerical comparison with the finite element results to be provided by us. Recent numerical approaches to this problem are those of Jeng and Chen [5-20] and of Jin and Liepa [5-21]. The first study uses a fundamental variational principle to derive a hybrid element method. The second uses an external boundary very close to the scatterer and follows its shape. Both the analytic and numerical results will be used as test cases for our parallel code.

Some recent studies in EM wave scattering include some new ideas which will need some examination on our part. One such approach, that of using boundary elements [5-22], is a flexible variant of the finite element method. Another approach, that of using on-surface boundary conditions, allows the external artificial boundary to shrink all the way to the surface of the scatterer [5-23]. This approach has been successfully used for simple geometric shapes, and one of its limitations has been the requirement of a convex scatterer.

In a recent study Peterson [5-16] has performed a comparative study of three different methods for TE-wave scattering from inhomogeneous dielectric cylinders. The first method is based on a volume discretization of a magnetic field integral equation [5-24]. The second method is a volume discretization of a differential equation formulation incorporating a "near field" radiation condition of the Bayliss-Turkel type [5-2], and is the method used by us in our test case. The third method is a hybrid procedure combining the discretization of a volume differential equation and a surface integral equation. Peterson finds that the methods incorporating differential equations have significant computational advantages in terms of storage over volume integral formulations. In addition, the hybrid approach requires less storage than the pure differential equation method, but may suffer from uniqueness problems for electrically large scatterers. It will be important to verify these conclusions in the context of parallel computation, especially comparing the hybrid method with the pure differential equation method currently used by us.

In summary, the purpose of first implementing the parallel 2-d code, which is now nearing completion, is to provide the essential background for formulating and implementing the more important, and considerably more complicated, three-dimensional scattering parallel code. Our intent is to consider specifically a coated conductor, where the coating can be a dielectric or magnetic material, possibly inhomogeneous, anisotropic, and lossy. With some of these cases currently available in a sequential, three-dimensional finite element code [5-25], our purpose is to combine these results with our currently gained insights of the parallel problem structure, leading to the formulation and construction of a parallel three-dimensional finite element scattering analysis code.

REFERENCES

[5-1] T. J. R. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[5-2] A. Bayliss and E. Turkel, "Radiation boundary conditions for wave-like equations," Commun. Pure and Appl. Math., Vol. 33, pp. 707-725, 1980; A. Bayliss, M. Gunzburger, and E. Turkel, "Boundary conditions for the numerical solution of elliptic equations in exterior regions," SIAM J. Appl. Math., Vol. 12, pp. 430-451, 1982.

[5-3] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, Solving Problems on Concurrent Processors, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[5-4] G. A. Lyzenga, A. Raefsky, and B. Nour-Omid, "Implementing finite element software on hypercube machines," Proc. Third Conf. on Hypercube Concurrent Computations and Applications, Pasadena, CA, January 19-20, 1988 (in press).

[5-5] C. F. Smith, A. F. Peterson, and R. Mittra, "A conjugate gradient algorithm of multiple incident electromagnetic fields," preprint, 1988.

[5-6] A. Jennings, Matrix Computation for Engineers and Scientists, John Wiley & Sons, New York City, NY, 1977; G. H. Golub and C. F. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MD, 1984; W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, Cambridge, U.K., 1986.

[5-7] T. K. Sarkar, K. R. Siarkiewicz, and R. F. Stratton, "Survey of numerical methods for solution of large systems of linear equations for electromagnetic field problems," IEEE Trans. Antennas and Propagation, Vol. AP-29, pp. 847-856, 1981.

[5-8] P. M. Van den Berg, "Iterative computational techniques in scattering based upon the integrated square error criterion," IEEE Trans. Antennas and Propagation, Vol. AP-32, pp. 1063-1071, 1984.

[5-9] A. F. Peterson and R. Mittra, "Method of conjugate gradients for the numerical solution of large-body electromagnetic scattering problems," J. Opt. Soc. Am., Vol. 2A, pp. 971-977, 1985; "Convergence of the conjugate gradient method when applied to matrix equations representing electromagnetic scattering problems," IEEE Trans. Antennas and Propagation, Vol. AP-34, pp. 1447-1454, 1986.

[5-10] B. Nour-Omid, A. Raefsky, and G. A. Lyzenga, "Solving finite element equations on concurrent processors," in Proc. Symp. on Parallel Computations and Their Impact on Mechanics, Boston, December 13-18, 1987; ASME, 1988.

[5-11] D. A. H. Jacobs, "The exploitation of sparsity by iterative methods," pp. 191-222, in I. S. Duff (ed.), Sparse Matrices and Their Uses, Academic Press, London, 1981.

[5-12] J. G. Lewis and R. G. Rehm, "The numerical solution of a non-separable elliptic partial differential equation by preconditioned conjugate gradients," J. Res. Nat. Bur. Stand., Vol. 85, No. 5, pp. 367-390, September-October, 1980.

[5-13] D. J. Evans (ed.), Preconditioning Methods: Analysis and Applications, Gordon and Breach, New York, 1983.

[5-14] A. F. Peterson, private communication, 1988.

[5-15] A. Kas and E. L. Yip, "Preconditioned conjugate gradient methods for solving electromagnetic problems," IEEE Trans. Antennas and Propagation, Vol. AP-35, No. 2, pp. 147-149, February 1987.

[5-16] A. F. Peterson, "A comparison of integral, differential and hybrid methods for TE-wave wave scattering from inhomogeneous dielectric cylinders," J. Electromagnetic Waves & Applications, in press, 1988.

[5-17] C. F. Smith, A. F. Peterson, and R. Mittra, "The biconjugate gradient method for electromagnetic scattering," submitted for publication, 1988.

[5-18] C. W. Helstrom, "Scattering from a cylinder coated with a dielectric material," in E. C. Jordan (ed.), Electromagnetic Theory and Antennas, Part 1, Pergamon Press, New York City, NY, 1963; T. C. K. Rao and M. A. K. Hamid, "G.T.D. analysis of scattering from a dielectric-coated conducting cylinder," IEEE Proc., Vol. 127, No. 3, pp. 141-153, June 1980; N. Wang, "Electromagnetic scattering from a dielectric-coated circular cylinder," IEEE Trans. Antennas and Propagation, Vol. AP-33, No. 9, pp. 960-963, September 1985.

[5-19] C.-C. Su, "Calculation of electromagnetic scattering from a dielectric cylinder using the conjugate gradient method and FFT," IEEE Trans. Antennas and Propagation, Vol. AP-35, No. 12, pp. 1418-1425, December 1987.

[5-20] S.-K. Jeng and C.-H. Chen, "On variational electromagnetics: theory and application," IEEE Trans. Antennas and Propagation, Vol. AP-32, No. 9, pp. 902-907, September 1984.

[5-21] J.-M. Jin and V. V. Liepa, "Application of hybrid finite element method to electromagnetic scattering from coated cylinders," IEEE Trans. Antennas and Propagation, Vol. 36, No. 1, pp. 50-54, January 1988.

[5-22] K. Yashiro and S. Ohkawa, "Boundary element method for electromagnetic scattering from cylinders," IEEE Trans. Antennas and Propagation, Vol. AP-33, No. 4, pp. 383-389, April 1985.

[5-23] G. A. Kriegsmann, A. Taflove, and K. S. Umashankar, "A new formulation of electromagnetic wave scattering using an on-surface radiation boundary condition approach," IEEE Trans. Antennas and Propagation, Vol. AP-35, No. 2, pp. 153-161, February 1987.

[5-24] A. F. Peterson and P. W. Klock, "An improved MFIE formulation for TE-wave scattering from lossy, inhomogeneous dielectric cylinders," IEEE Trans. Antennas Propagation, Vol. AP-36, No. 1, pp. 45-49, January 1988.

[5-25] "EM-TRANAIR; A Computer Program for the Solution of Maxwell's Equations in Three Dimensions: Vol. 1, Theory Manual," Boeing Report, Seattle, WA, July 1987.

SECTION VI

TEST CASE: SCATTERING FROM A CONDUCTING SPHERE

A. DESCRIPTION OF THE TEST CASE

In this section we present results from both the parallel Numerical Electromagnetics Code (NEC) and the parallel Finite Difference Time Domain (FDTD) code for the test case. The test case is scattering from a perfectly conducting sphere. We compare two components of the near scattered fields, two far electric field components, and the bistatic radar cross section (RCS). We also compare these results to the analytic solution.

The test case is a conducting sphere with a diameter equal to the wavelength of the incident plane wave radiation. The incident plane wave has a frequency equal to 300 MHz. The incident plane wave has a wavelength of 0.99933 m and a wave number of 6.2874 rad/m. These parameters produce a sphere with a radius of 0.49967 m. Ka, the wave number of the incident field times the radius of the spherical scatterer, equals 3.1415927.

Since the scatterer is symmetric with respect to the origin, we arbitrarily choose the incident field to propagate from the -y direction to the +y direction. The incident electric field points in the +z direction with a magnitude equal to 1.0 V/m. The incident magnetic field points in the +x direction with a magnitude equal to 1/376.7303 A/m.

B. RESULTS AND COMPARISONS

1. Test Case Using the Finite Difference Time Domain Code

To model the sphere in the FDTD code, we break up the sphere into conducting cubic cells. Sixty cells span the diameter of the sphere. The FDTD code replaces the smooth spherical surface with protruding cubic shapes. We also model the empty space surrounding the conducting sphere with these cubic cells. The program marks unit cells with centers within the radius of the sphere as part of the scatterer. Unit cells with centers outside the sphere radius are part of the vacuum. We use 80 x 80 x 80 unit cells to discretize the computation space. We use 60 x 60 x 60 cells to enclose the spherical

scatterer. With these parameters, the unit cell size is 0.016656 x 0.016656 x 0.016656 m and the global lattice size is 1.3324 x 1.3324 x 1.3324 m.

For this problem the Courant condition determines 120 iterations per cycle of the incident field. Therefore, time increments 0.02778 nsec per time step iteration. To determine the convergence of the FDTD code, we observe the monostatic RCS. This value is the radar cross section observed from the direction of the plane wave source. Figure 6.1 shows the value of the monostatic RCS as a function of the iteration count. From this plot, we observe that after ~900 iterations the near scattered fields achieve steady state. When the near fields achieve steady state, the value of the monostatic RCS is approximately 0.677 m^2.

2. Test Case Using the Numerical Electromagnetics Code

To model the spherical scatterer in NEC, we build an eighth of a unit sphere and then use the scaling option in NEC to create the proper radius and the symmetry option in NEC to build the complete scatterer. The scaling option multiples all lengths by a user-specified constant value. The symmetry option reflects the eighth of a unit sphere through three planes passing through the origin and perpendicular to the Cartesian coordinate axes. We enter the scaling option using the GS card in NEC. We use the GX card to enter symmetry options. For the eighth of the spherical scatterer, we use arbitrarily shaped surface patches to construct the object. We constructed this scatterer using the structure editor module in the Electromagnetic Interactive Analysis Workstation (EIAW). Figure 7.5 in Section VII shows the construction commands for PATRAN. Figure 7.6 shows the arrangements of the sixty surface patches for the eighth of the spherical scatterer. Figure 7.8 shows a portion of the geometry section of a NEC input file specifying the sphere [6-1].

We use the card editor module of the EIAW to place the incident field excitation in the NEC input file. The parameters for the direction and polarization of the incident field are the following: θ, the angle off the z axis, equals 90° degrees; ϕ, the angle off the x axis, equals 270°; and η, the angle off the $\hat{\theta}$ vector, equals 180°. We used this same field in the FDTD run.

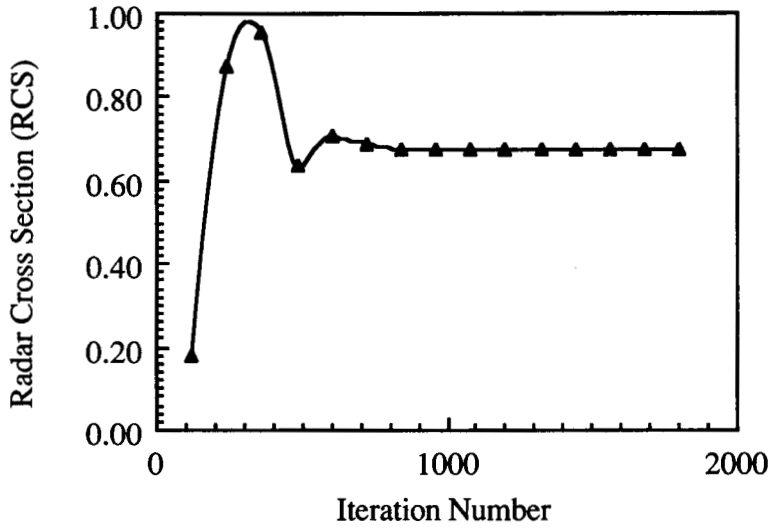


Figure 6.1. Plot of the monostatic radar cross section versus the number of iterations. Each iteration is 0.0278 nsec.

Compared with the 0.677 m^2 value for the monostatic RCS from the FDTD code, the NEC code gives a value of -1.74 dB or 0.6699 m^2.

3. Test Case Using the Exact Solution

We attribute the exact solution for the spherical scatterer to D. Mie [6-2]. The solution uses spherical wave functions and boundary value techniques to find the terms of the resultant Mie series for the scattered field. A detailed mathematical discussion of the solution can be found in Ruck et al. [6-3] or Bowman et al. [6-4].

Dr. Sembiam Rengarajan from California State University Northridge furnished a FORTRAN code for the evaluation of the series. The code only generates scattered near fields. The code was tested by comparison with near field plots in [6-4].

Compared with the 0.677 m^2 value from the FDTD code and with the 0.6699 m^2 value from the NEC code, the monostatic RCS from exact solutions is 0.6667 m^2 [6-3].

The percent difference between the exact RCS and the RCS from the NEC code is 0.47%. The percent difference between the exact RCS and the RCS from the FDTD code is 1.03%.

4. Contour Plots of Different Fields for the Spherical Scatterer

Thce following figures shows sets of plots obtained from the NEC, FDTD, and analytic codes. Figures produced by the NEC code and FDTD code appear together. Figures produced by the analytic code follow the NEC and FDTD results. Figures 6.2 through 6.7 show contour plots for the magnitude and phase of the z component of the scattered electric field. Figures 6.8 through 6.13 show contour plots for the magnitude and phase of the x component of the scattered magnetic field. These near field values occur in a plane in front of the spherical scatterer at y = -0.6162556 m. Figures 6.14 and 6.15 give contour plots of E_θ and E_ϕ for NEC and FDTD codes only. These fields do not include the radial dependent term e^{ikr}/r. K is the wave number of the incident field and r is the radial distance from the origin at the center of the spherical scatterer. Figure 6.16 shows the bistatic RCS for NEC and FDTD codes.

Because the fields produced by the three codes may differ in phase by some constant angle, we freely shift all phase values produced by a single code by the same constant. This shift produces plots for easier comparison. For the following near field plots we shifted the phase values from NEC by -27° and the phase values from the analytic code by 155°.

5. Discussion of Results

NEC compares well with the analytic code in both magnitude and phase for the near scattered fields. However, the FDTD code has slightly differing field values for magnitude and phase. We can trace the difference in the near fields to the cubic approximation to the true spherical scatterer.

We can look at the percent difference of a few field components for the three codes. NEC gives 0.20384 V/m for the magnitude of the z component of the electric field at x = -0.61625 m, y = -0.61625 m, and z = -0.60793 m. This point is the lower-leftmost value in the NEC contour plot of Figure 6.2. FDTD gives 0.20397 V/m for the same magnitude.

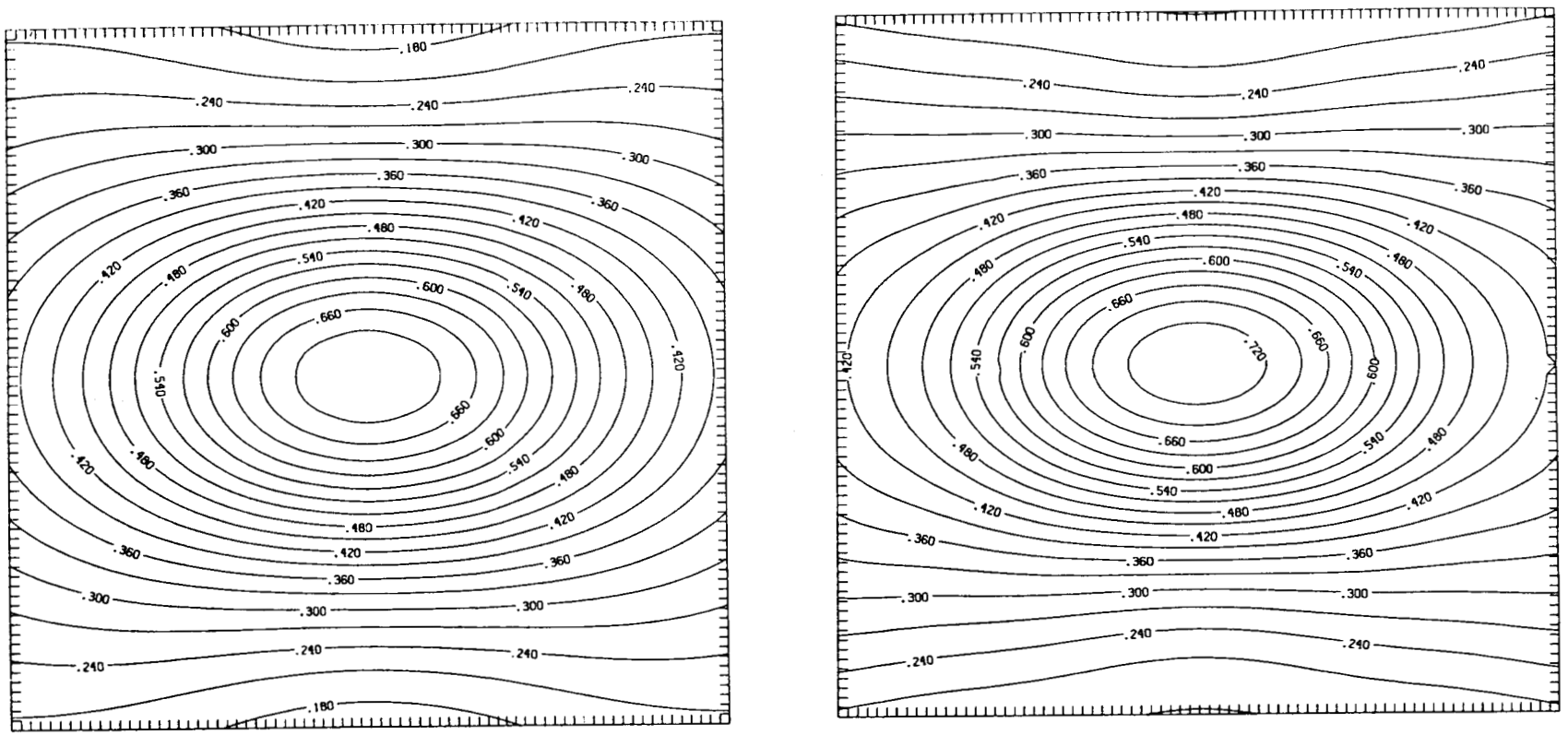


Figure 6.2. Contour plots, produced by the NEC code (left) and the FDTD code (right), of the magnitude of the z component of the scattered electric field in a plane in front of the sphere scatterer. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

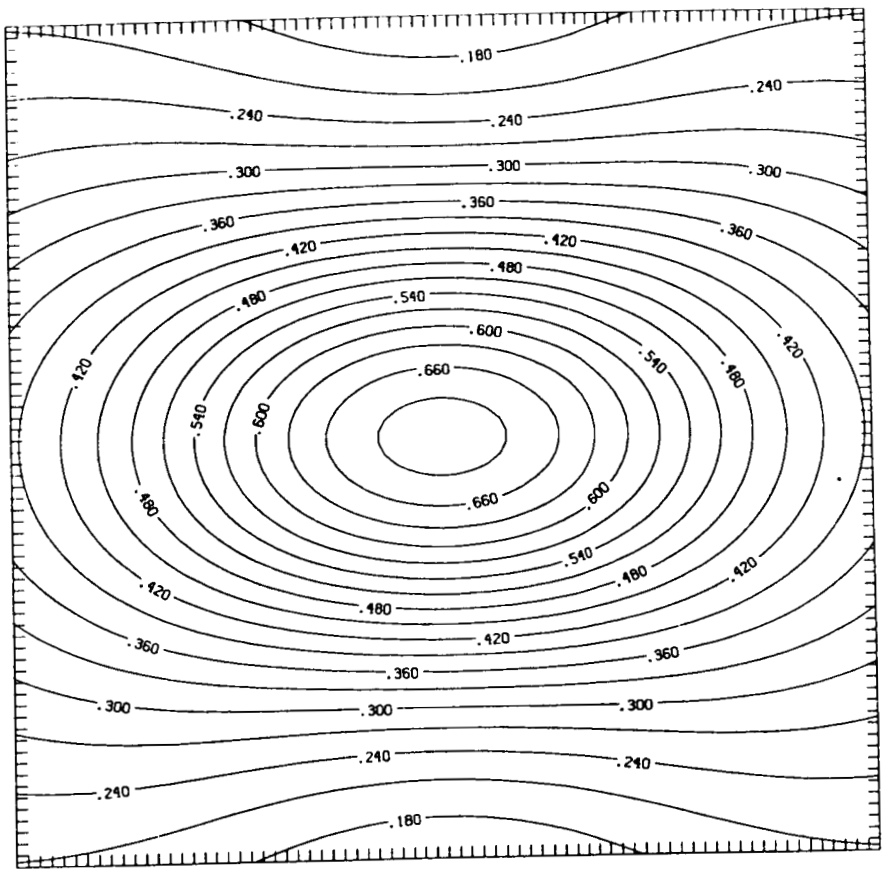


Figure 6.3. Contour plot, produced by the exact solution code, of the magnitude of the z component of the scattered electric field in a plane in front of the sphere scatterer. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

Figure 6.4. Contour plots, produced by the NEC code (left) and the FDTD code (right), of the phase of the z component of the scattered electric field in a plane in front of the sphere scatterer. The phase angles range from 0 to -90 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

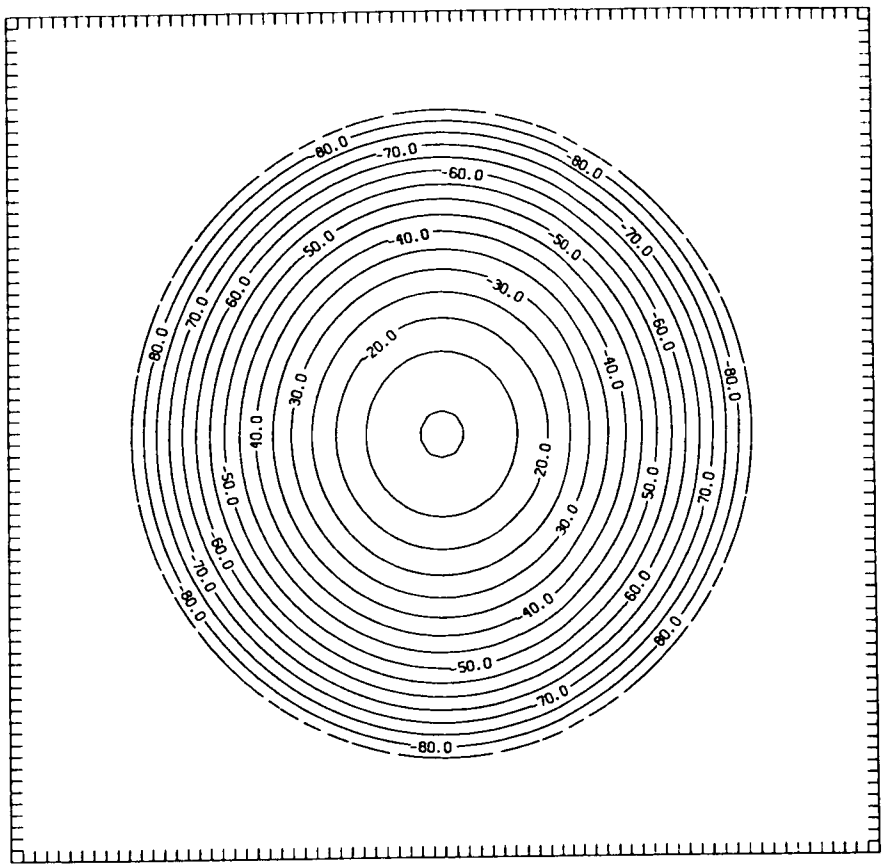


Figure 6.5. Contour plot, produced by the exact solution code, of the phase of the z component of the scattered electric field in a plane in front of the sphere scatterer. The phase angles range from 0 to -90 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

Figure 6.6. Contour plots, produced by the NEC code (left) and the FDTD code (right), of the phase of the z component of the scattered electric field in a plane in front of the sphere scatterer. The phase angles range from -90 to -180 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

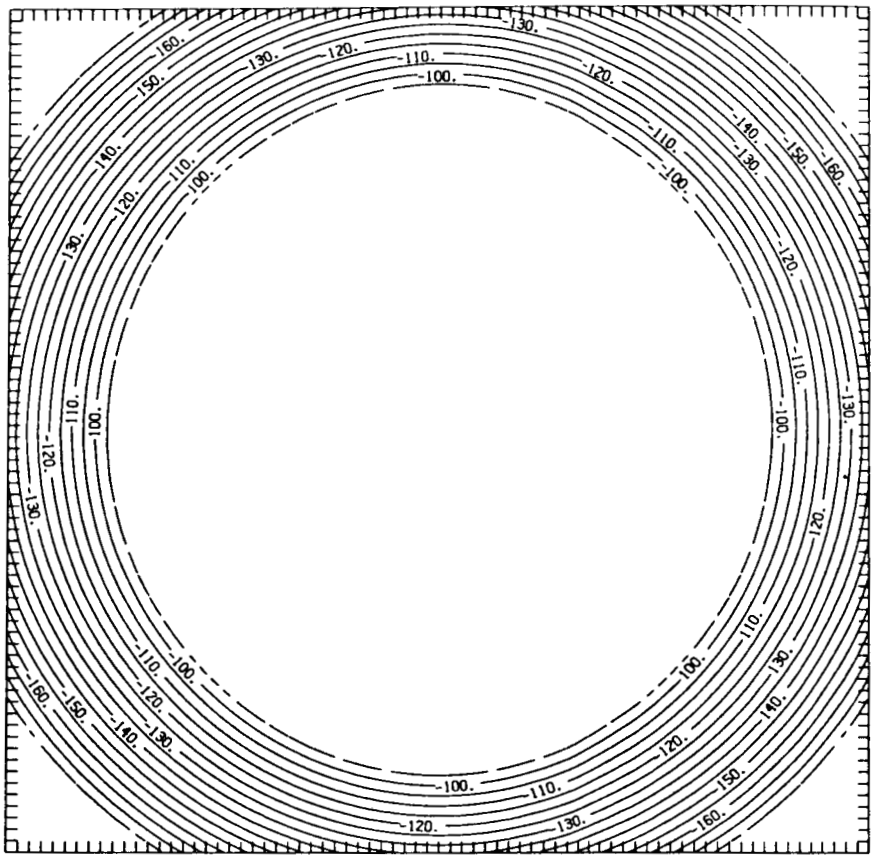


Figure 6.7. Contour plot, produced by the exact solution code, of the phase of the z component of the scattered electric field in a plane in front of the sphere scatterer. The phase angles range from -90 to -180 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

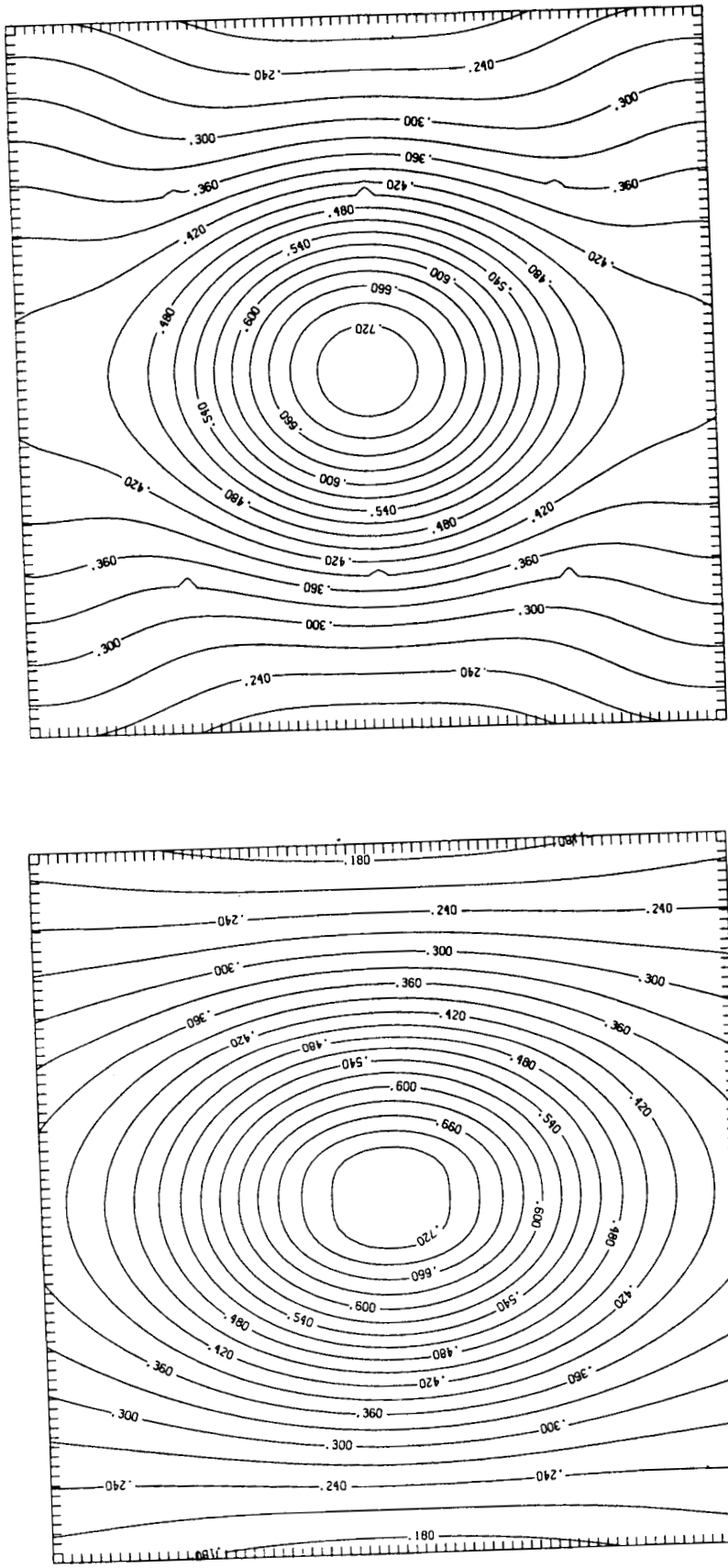


Figure 6.8. Contour plots, produced by the NEC code (left) and the FDTD code (right), of the magnitude of the x component of the scattered magnetic field in a plane in front of the sphere scatterer. We multiply the actual magnitude of the field by 376.7303. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

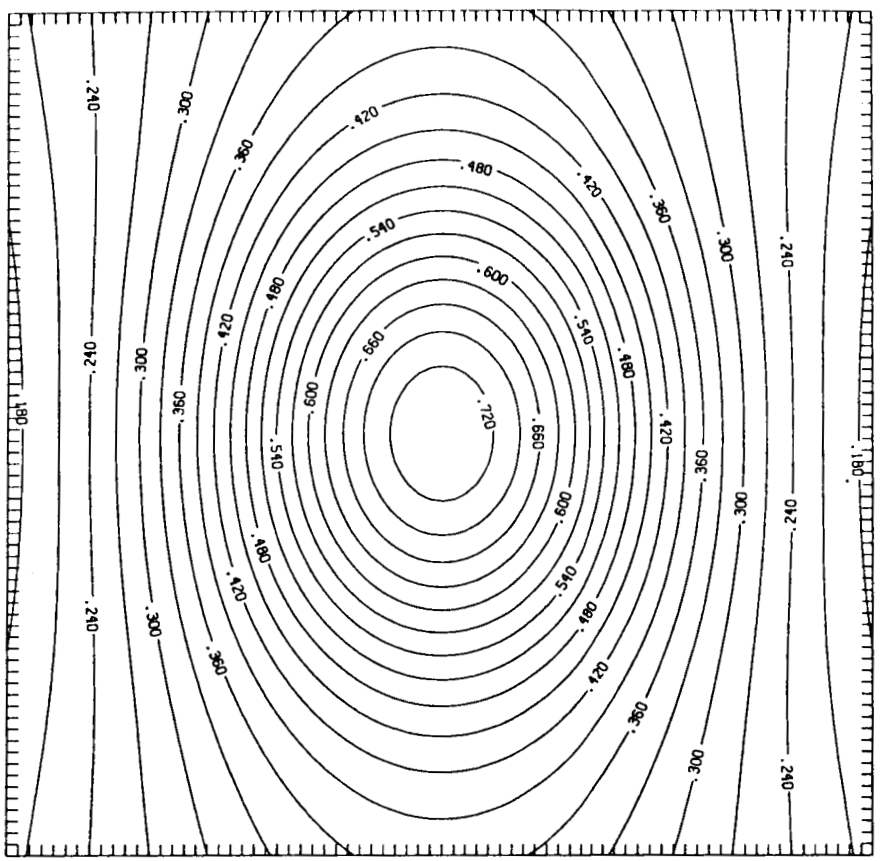


Figure 6.9. Contour plot, produced by the exact solution code, of the magnitude of the x component of the scattered magnetic field in a plane in front of the sphere scatterer. We multiply the actual magnitude of the field by 376.7303. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

Figure 6.10. Contour plots, produced by the NEC code (left) and the FDTD code (right), of the phase of the x component of the scattered magnetic field in a plane in front of the sphere scatterer. The phase angles range from 0 to +90 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

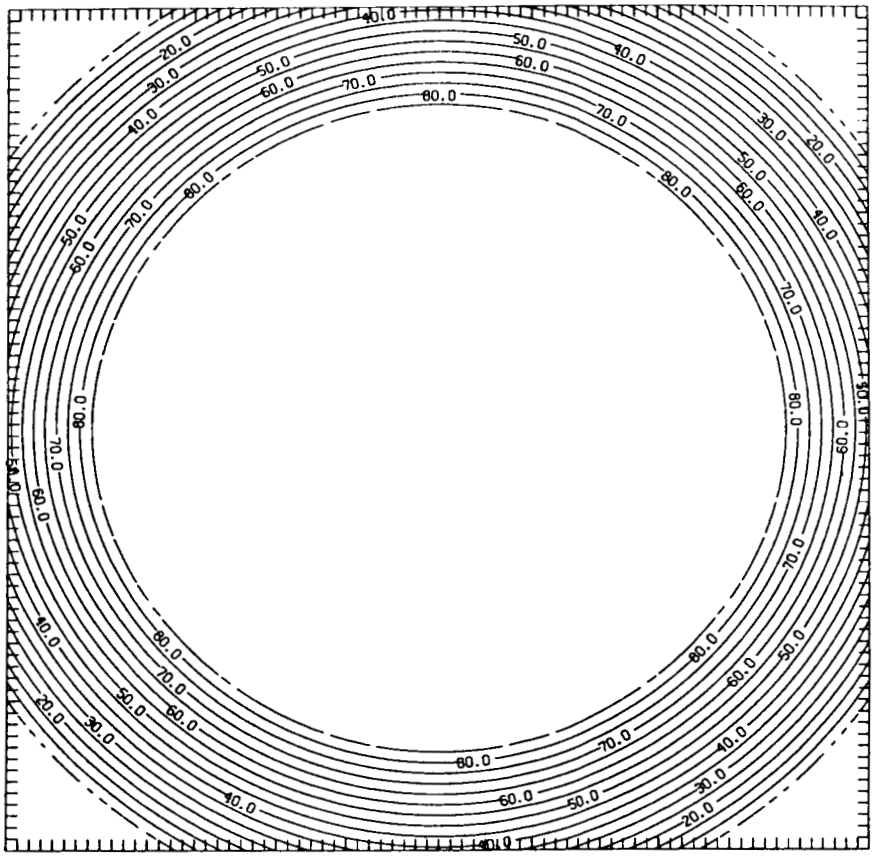


Figure 6.11. Contour plot, produced by the exact solution code, of the phase of the x component of the scattered magnetic field in a plane in front of the sphere scatterer. The phase angles range from 0 to +90 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

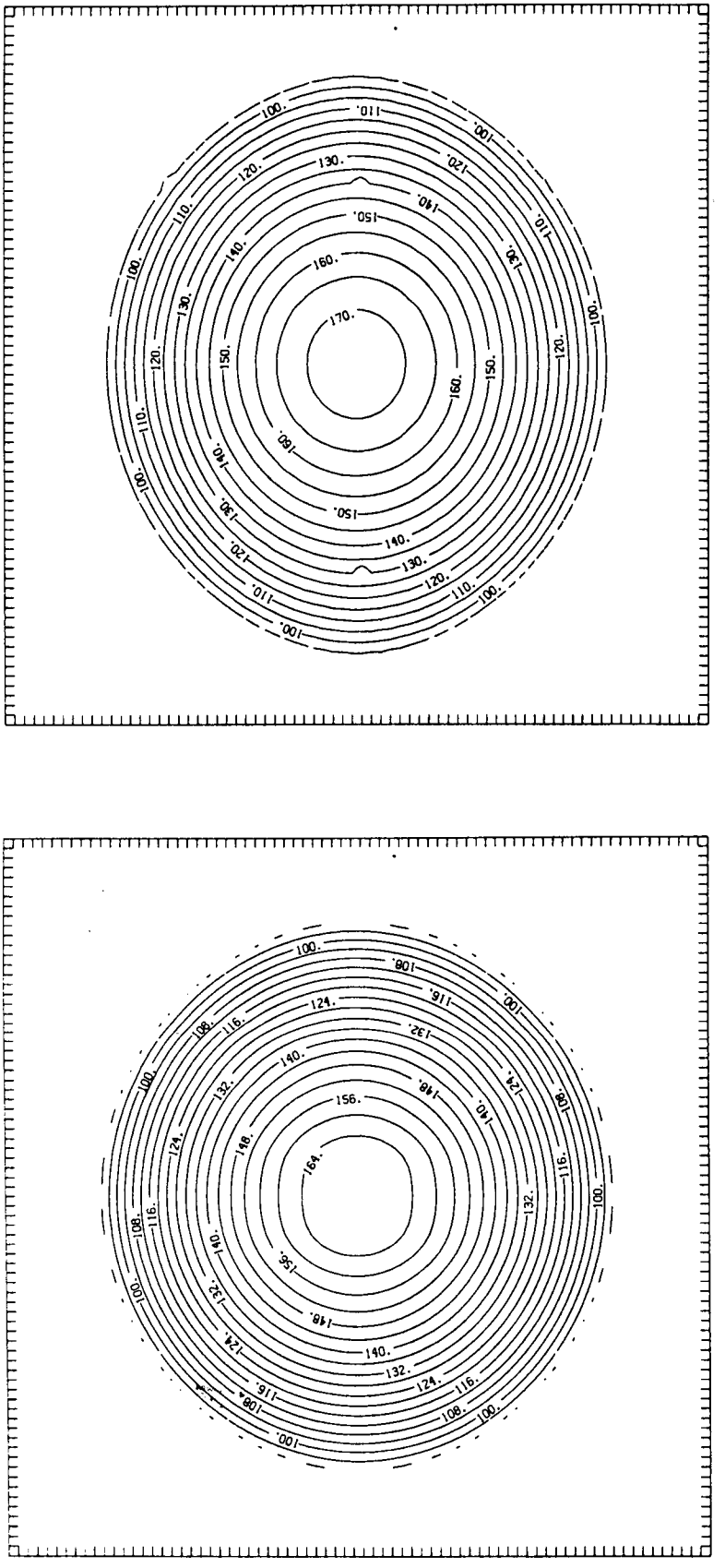


Figure 6.12. Contour plots, produced by the NEC code (left) and the FDTD code (right), of the phase of the x component of the scattered magnetic field in a plane in front of the sphere scatterer. The phase angles range from 90 to 180 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

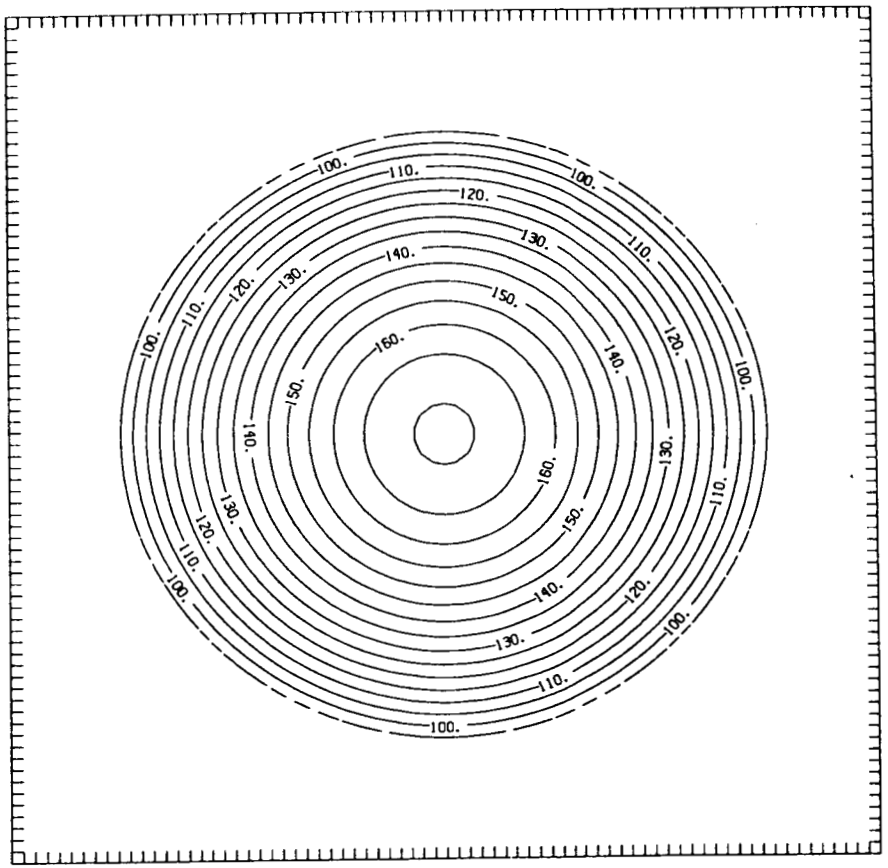


Figure 6.13. Contour plot, produced by the exact solution code, of the phase of the x component of the scattered magnetic field in a plane in front of the sphere scatterer. The phase angles range from 90 to 180 degrees. The coordinates of this plane are as follows: x equals -0.6162556 to +0.6162556 m in steps of 0.0166556 m, y equals -0.6162556 m, and z equals -0.60792778 to +0.60792778 m in steps of 0.0166556 m. We plot the x coordinates along the horizontal axis and the z coordinates along the vertical axis.

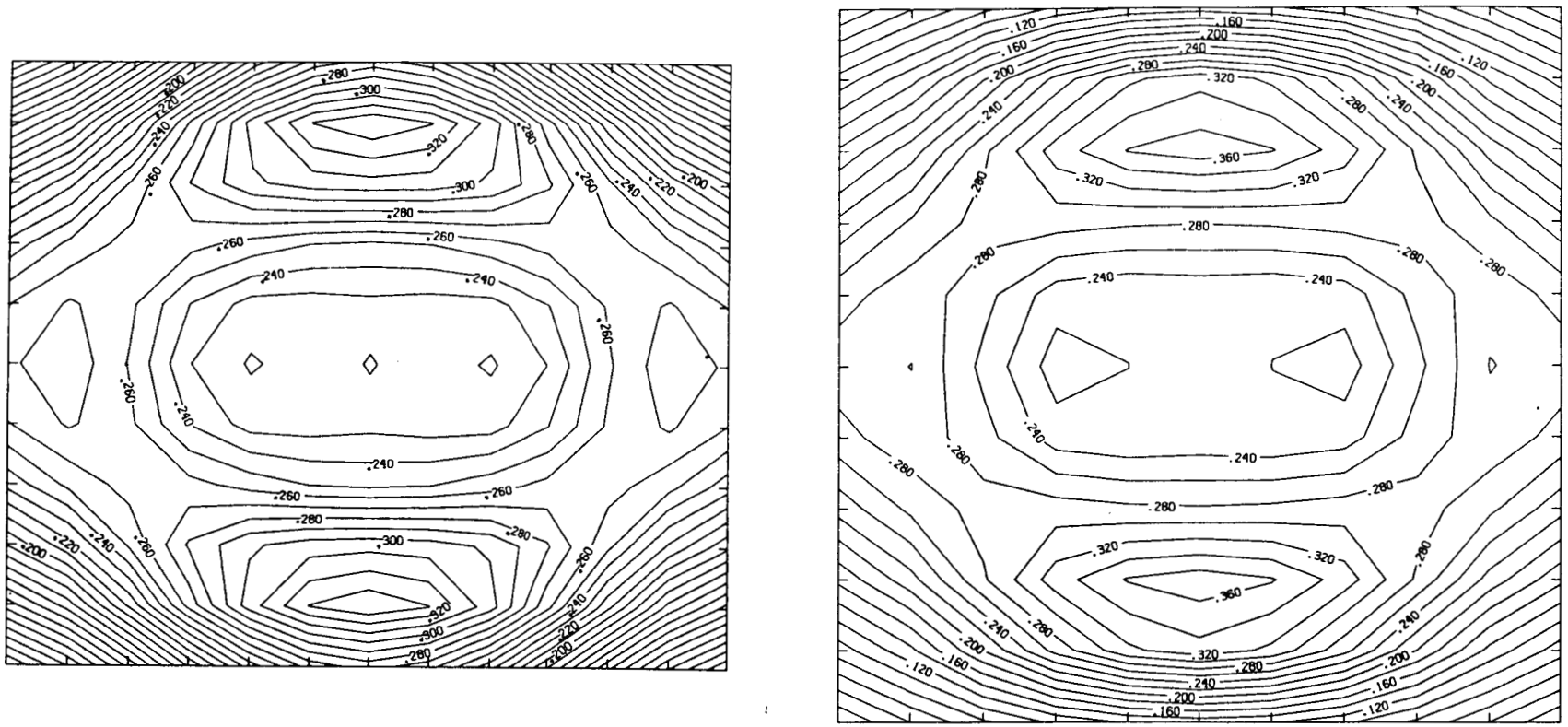


Figure 6.14. Contour plots of the magnitude of the θ component of the scattered far field. The magnitude excludes the e^{ikr} / r factor. K is the wave number of the incident field and r is the distance from the origin. We plot the θ angle along the vertical axis and the ϕ angle along the horizontal axis. We produced the plot on the left using NEC. In this plot, the ϕ angle ranges from 180° to 360° in steps of 15°, and the θ angle ranges from 15° to 165° in steps of 15°. We produced the plot on the right using FDTD. In this plot, the ϕ angle ranges from 180° to 360° in steps of 18°, and the θ angle ranges from 0° to 180° in steps of 18°.

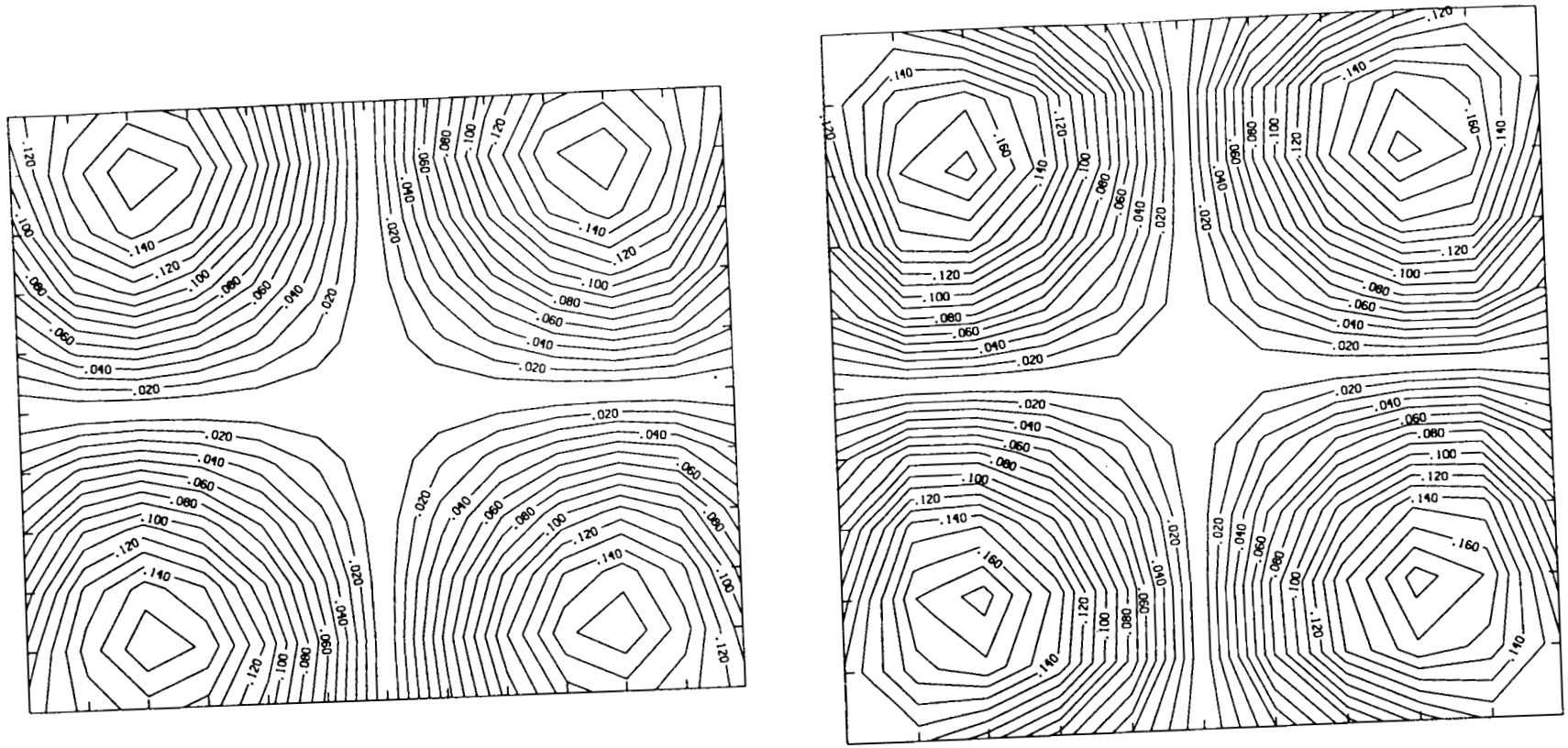


Figure 6.15. Contour plots of the magnitude of the ϕ component of the scattered far field. The magnitude excludes the e^{ikr} / r factor. K is the wave number of the incident field and r is the distance from the origin. We plot the θ angle along the vertical axis and the ϕ angle along the horizontal axis. We produced the plot on the left using NEC. In this plot, the ϕ angle ranges from 180^o to 360^o in steps of 15^o, and the θ angle ranges from 15^o to 165^o in steps of 15^o. We produced the plot on the right using FDTD. In this plot, the ϕ angle ranges from 180^o to 360^o in steps of 18^o, and the θ angle ranges from 0^o to 180^o in steps of 18^o.

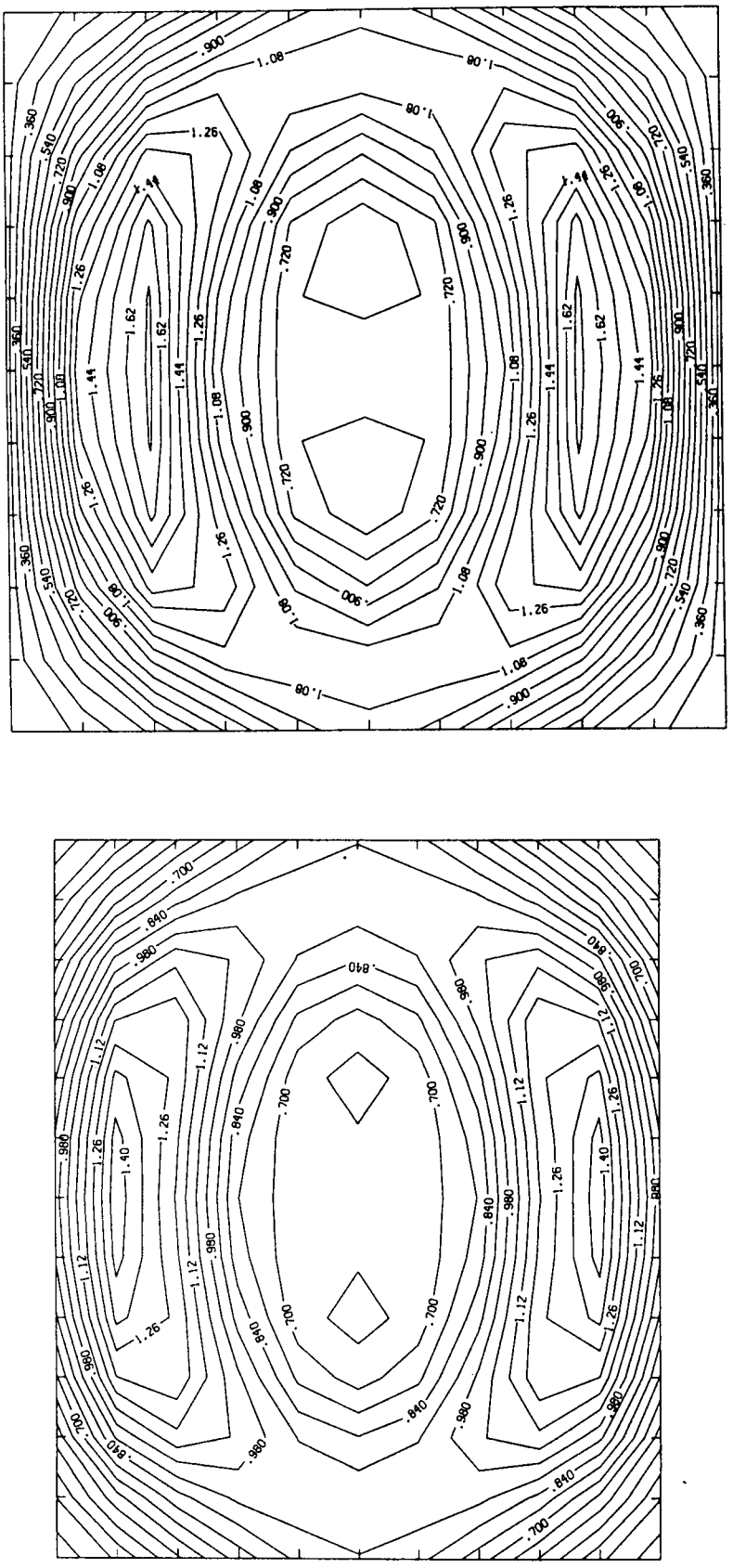


Figure 6.16. Contour plots of the radar cross section (RCS). We plot the θ angle along the vertical axis and the ϕ angle along the horizontal axis. We produced the plot on the left using NEC. In this plot, the ϕ angle ranges from 180° to 360° in steps of 15°, and the θ angle ranges from 15° to 165° in steps of 15°. We produced the plot on the right using FDTD. In this plot, the ϕ angle ranges from 180° to 360° in steps of 18°, and the θ angle ranges from 0° to 180° in steps of 18°.

This point is the lower-leftmost value in the FDTD contour plot of Figure 6.2. The analytic code gives 0.20805 V/m for the same magnitude. This point is the lower-leftmost value in the analytic solution contour plot of Figure 6.3. The percent difference between the NEC and analytic value is 2.0%. The percent difference between the FDTD and analytic value is 2.0%.

We can also look at the value of the z component of the near scattered field at x = 0.0, y = -0.61625, and z = -0.0083. This point is near the center of the contour plots of Figures 6.2 and 6.3. The z component attains its maximum value on the contour plot at this point. The values from the NEC, FDTD, and analytic codes for the z component at this point are as follows: 0.71693 V/m, 0.74393 V/m, and 0.70390 V/m. The percent difference between the NEC and analytic value is 2.0%. The percent difference between the FDTD and analytic value is 5.7%.

Intuitively, the sphere, modeled in the FDTD code, scatters more of the incident field towards the source than the sphere modeled in NEC. The bistatic RCS plots of Figure 6.16 also confirm this observation. The RCS values for varying theta and phi from the FDTD code are consistently greater than the RCS values from the NEC code.

From the CPU times, we also conclude that the FDTD code is less efficient than the NEC code in finding the scattered fields from the sphere. However, the designers of NEC explicitly wrote the code to deal with steady-state analysis of conducting structures. FDTD, on the other hand, can do either a steady-state or a transient analysis of either perfectly conducting scatterers or inhomogeneous dielectrics.

REFERENCES

[6-1] G. J. Burke and A. J. Poggio, Numerical Electromagnetics Code (NEC) - Method of Moments, Lawrence Livermore National Laboratory, Livermore, CA, 1981.

[6-2] J. A. Stratton, Electromagnetic Theory, McGraw-Hill Book Company, New York, 1941.

[6-3] G. T. Ruck, D. E. Barrick, W. D. Stuart, and C. K. Krickbaum, Radar Cross Section Handbook, Vol. 1, Plenum Press, New York City, NY, 1970.

[6-4] J. J. Bowman, T. B. A. Senior, and P. L. E. Uslenghi, Electromagnetic and Acoustic Scattering by Simple Shapes, North Holland Publishers, Amsterdam, the Netherlands, 1969.

SECTION VII

THE ELECTROMAGNETIC INTERACTIVE ANALYSIS WORKSTATION

A. OVERVIEW

With the Numerical Electromagnetics Code (NEC) and the Finite Difference Time Domain (FDTD) Code available on the Mark III Hypercube, there is a need to ease the user interaction with these programs. Software tools must exist for the user of these programs to interactively create input files, remotely execute these analysis codes, and graphically display the results of the analysis. To this end, we created an Electromagnetic Interactive Analysis Workstation (EIAW) on a Sun Microsystem 3/160 workstation. We integrated the NEC code into the workstation environment.

As the complexity of scattering objects increases, NEC and other analysis codes require a graphics input generator to pictorially create the scattering or radiating object. A user also may want to visually verify the correctness of an existing structure specification. Many software packages exist that allows the user to create and view a three-dimensional object and produce a output file with structure information. However, after considering many Computer Aided Design (CAD) and Finite Element (FEM) packages, we settled on PATRAN-PLUS from PDA Engineering. This graphics package (GPK) is one of the modules to aid the interactive creation of input files.

Because we cannot expect any off-the-shelf graphics software to create a correctly formatted NEC input file, we must construct a program called a translator that allows GPK and NEC to communicate. Our particular translator creates the geometry portion of the NEC input file from information stored in GPK's output files. PATRAN's output files are the patch area file and the application-independent neutral file. This translator also creates PATRAN neutral files from the geometry portions of the NEC input file. This translator package is the next module to aid the interactive creation of input files.

Because the user cannot graphically specify with GPK all the input cards in a typical NEC input deck, we created an input editor. A card in an NEC input file is a one-line entry beginning with a two-column header and followed by a set of numerical fields. Typical headers include: SP for surface patch, GW for a set of wires, and EX for excitation on a structure. Examples of cards appear below. This editor, called a card editor, allows

the user to add or delete additional cards to an existing NEC input file. The editor gives the user a full list of input cards, prompts the user for the appropriate entry in each field of a card, and checks each field for the correct syntax. Together with GPK and the translator, the card editor allows the user to create a complete NEC input file.

Because the EIAW software exists on a Sun workstation and the host to the Hypercube is a Counterpoint System XIX workstation, we must create a run control package. Starting a Hypercube application is usually an interactive process with the operator logged onto the host of the Hypercube. Therefore, the NEC code or any other application must sit on the Counterpoint. However, the EIAW modules exist on a Sun connected to the Counterpoint via Ethernet. We use the UNIX system and networking commands on the Sun to allow users to remotely submit hypercube programs from the workstation environment. Figure 7.1 shows the connections between the Sun 3/160; the Hypercube control processor (CP), a Counterpoint System XIX; and the Mark III Hypercube. The external hard disks hanging off the eight-node subcubes are enhancements which are important to Numerical Green's Function solutions for NEC.

Because electromagnetic codes produce pages of numerical output for the near and far fields, the user must have a graphics tool to visualize these fields produced by the scattering or radiating structure. For the designer, the ability to graphically analyze the output from a particular object will greatly aid in the interactive modification of the scattering structure. GPK also will serve to display the results from the analysis code.

One type of graphical display is colored contour plots. The user plots values of a quantity as a function of coordinates. PATRAN indicates the value by a corresponding color from a spectrum displayed on a color bar to the side of the PATRAN graphics screen.

PATRAN requires two files to create colored contour plots. The first file is a neutral file. It contains the specification of the grid on which the user superimposes the field values at the grid node points. The second file, called a results file, contains the node IDs and columns of field values associated with a particular node. Because we can assign several different field values to the same node number, we can create several plots of these distinct fields on the same console screen by requesting PATRAN to plot the various columns. To produce these two files, we use an output generation program utilizing some of NEC's output routines and new routines to properly format the different field values.

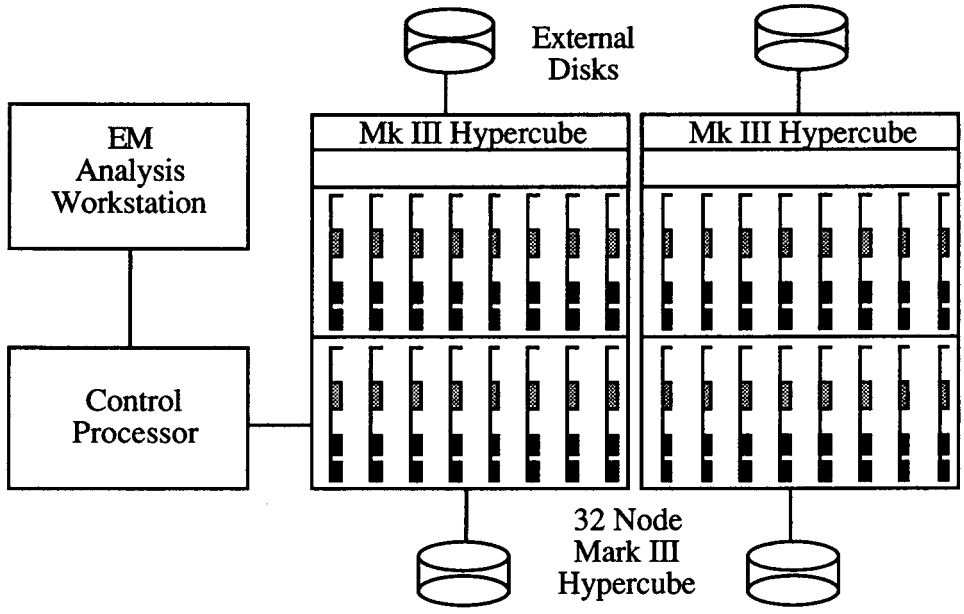


Figure 7.1. Diagram of the Electromagnetic Interactive Analysis Workstation, Hypercube Control Processor, and Mark III Hypercube

As the number of software modules increases, we want the user to freely move around in the workstation environment without the need to memorize complicated computer commands. Therefore, we integrated the above modules in a menu and prompt driven environment. In this environment, the user can move from one analysis tool to the next using a mouse device. Figure 7.2 illustrates possible analysis paths between the different modules.

B. MENU ENVIRONMENT

The Sun 3/160 comes with a software package, called SunView, to create custom windows and menus on the console screen [7-1]. Windows are distinct rectangular workspaces. Windows may also contain smaller subwindows. Depending on the type of subwindows, the user can run text-editing programs or application programs, plot graphics, or execute UNIX system commands within particular subwindows. A menu is a list of command options. Choice of a particular option with a pointing device, such as a keyboard or mouse device, initiates the execution of the command. Work within a

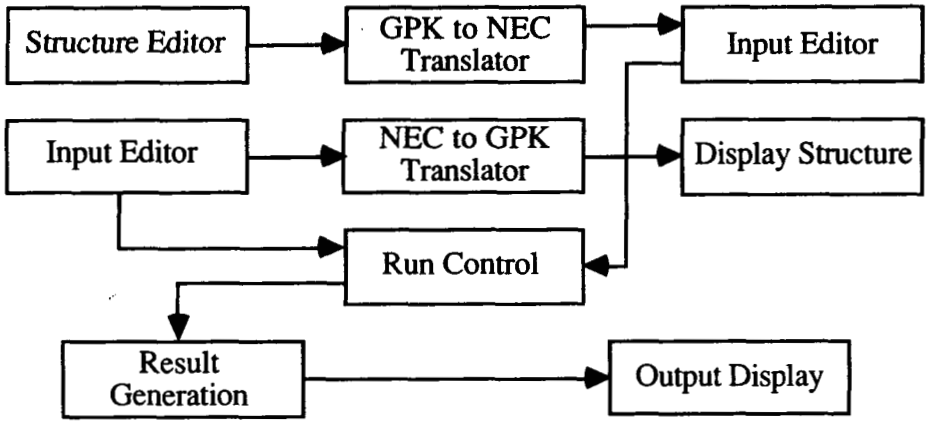


Figure 7.2. Analysis paths between modules of the Electromagnetic Interactive Analysis Workstation

subwindow may involve the use of custom menus to ease user interaction. Using these tools, we created custom windows, subwindows, and menus for the modules of EIAW.

When the EIAW is invoked, a large window appears on the console. This window contains one subwindow of menus, messages, and prompts and a second subwindow for entering UNIX system commands. Figure 7.3 shows this first window. The menu orders all the separate modules into the following categories: Setup Menu, Run Menu, and Output Menu. Choice of any of the menu options with the mouse device brings up a lower level menu.

We first discuss the Setup Menu and the EIAW modules it launches. We will discuss the Run Menu and Output Menu later in this section. Figure 7.4 shows the contents of the Setup Menu. Choice of Select Directory brings up other menu items that allow the user to move around in the file system directory hierarchy. Choice of Card Editor invokes the interactive card editor program. Choice of Display Structure or Structure Editor invokes GPK for object display or creation. Choice of File Translation invokes the NEC to GPK and GPK to NEC translator. Return moves back up to the Main Menu. We discuss these modules in detail below.

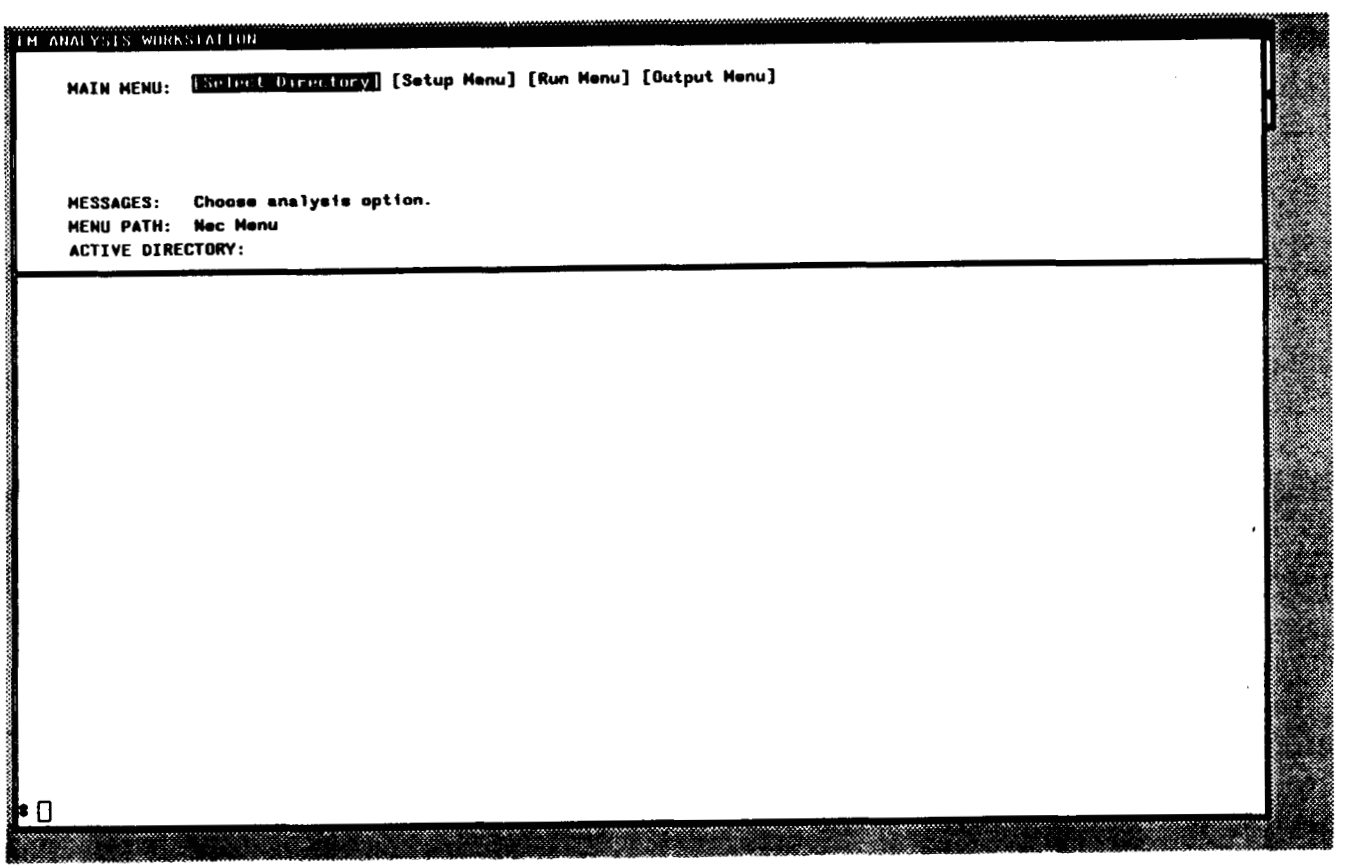


Figure 7.3. Main window of the EIAW showing the Main Menu

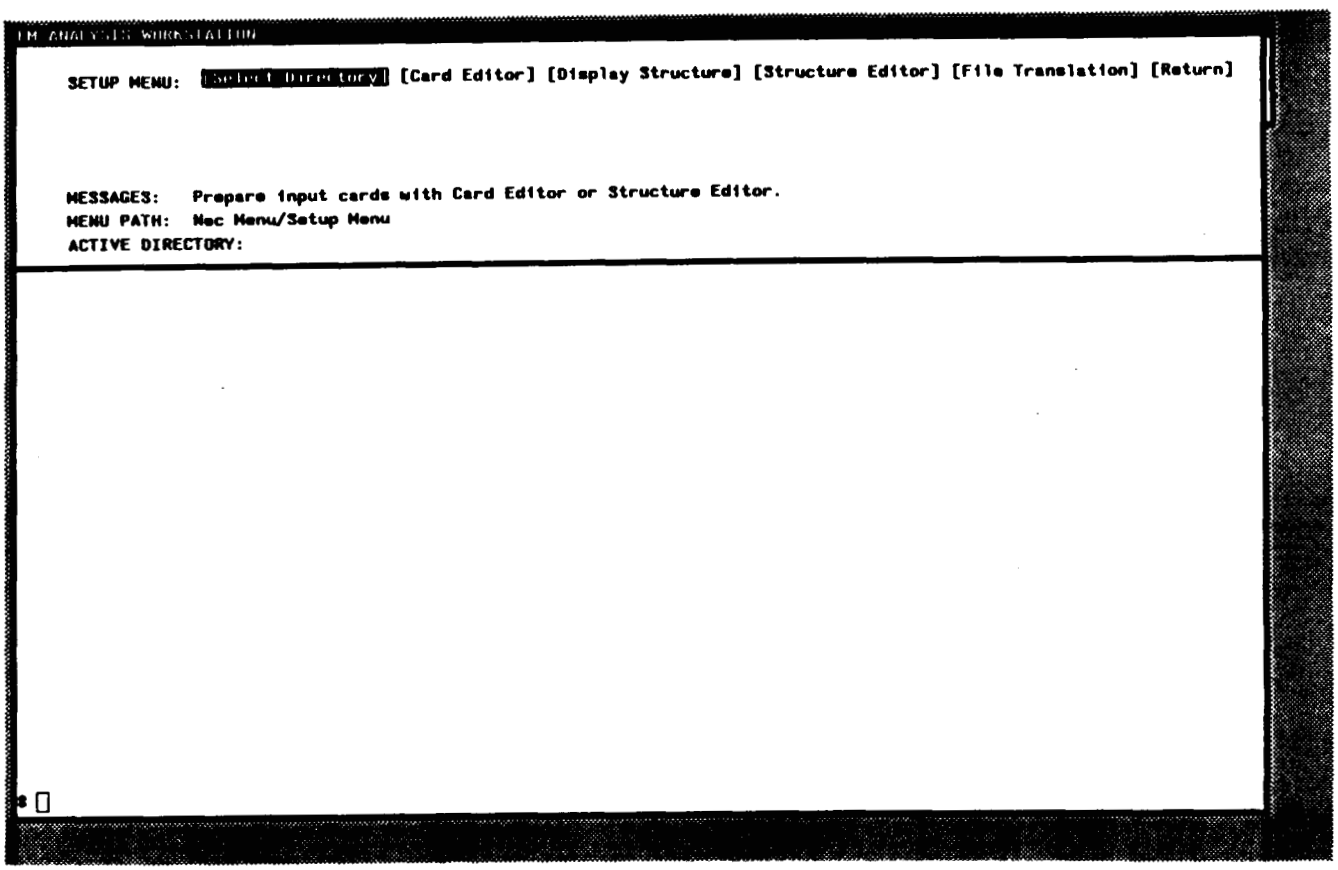


Figure 7.4. Main window of the EIAW showing the Setup Menu

ORIGINAL PAGE IS
OF POOR QUALITY

C. STRUCTURE EDITOR

A structure editor is software capable of accepting interactive three-dimensional graphics commands from an operator and creating a text or binary file containing the numerical description of the object. For NEC these three-dimensional objects may be flat or curved patches or straight wires. The structure editor also must present rotated and translated views of the object for verification of correctness. Because a finite element modeler embodies all the above features, we incorporated PATRAN as the structure editor in the EIAW. In the following paragraphs, we will briefly explain PATRAN's modeling method and then present a few simple examples [7-2]. The user should refer to the graphics package's user manual for a full description of its capabilities.

1. Modeling Commands

Modeling a scattering or radiating structure in PATRAN goes through the following sequence: geometric creation of a complex three-dimensional object, superimposing a mesh of finite element nodes on the geometric structure, connecting the nodes to produce finite element shapes, and assigning property numbers to each finite element shape. We can associate numbers or text strings to groups of finite element shapes. A property number identifies the set of numbers or text strings.

Geometric creation of a complex three-dimensional object requires PATRAN commands of the following form:

> Object Type, New Object ID, Generation Method, Generation Parameters, Original Object

Object Type is one of Grid (GR), a point object; Line (LI), a one-dimensional object; Patch (PA), a two-dimensional object; or Hyperpatch (HP), a three-dimensional object. The tangent vectors to the surface of these objects may vary with position. New Object ID is either a wild card character, #, or a list of numbers identifying the newly created objects. Generation Method specifies the method for creating the new objects. We give examples of generation methods in examples below. Generation Parameters give additional information required for the specific method of creating the new object. Original Object gives the IDs

of already existing objects from which the user creates new objects by rotation, translation, or duplication operations.

Creating finite element nodes on the structure requires a command of the following form:

GFEG, Object ID, , N1/N2/N3/r1/r2/r3

PATRAN requires the GFEG characters. Object ID is the number of the object created by the above geometric creation commands. N1 is the number of finite element nodes in the first parametric direction of the object. R1 is the ratio of the distance between the last pair of nodes and the first pair of nodes. If r1 is not unity, the distances increment or decrement in a geometric progression from the first pair of nodes to the last pair of nodes. If the object is multidimensional, N2 and N3, respectively, specify the number of finite element nodes in the other two parametric directions; r2 and r3 specify the geometric progression of distances between nodes in the other parametric directions.

Creating finite element shapes connecting nodes on the structure requires a command of the following form:

CFEG, Object ID, Finite Element Type

PATRAN requires the CFEG characters. Object ID is the number of the object created by the geometric creation commands. Finite Element Type specifies the type of finite element connecting nodes. Finite element types include BAR, a one-dimensional element connecting two nodes; TRI, a two-dimensional element connecting three nodes; and QUAD, a two-dimensional element connecting four nodes.

Assigning property IDs to finite element shapes requires a PATRAN command of the following form:

PFEG, Object ID, Finite Element Type, Numerical Values Or Text List

PATRAN requires the PFEG characters. Object ID is the number of the object created by the geometric creation commands which we discussed above. Finite Element Type specifies the type of finite element shape. This type must match the type specified in the CFEG command. Numerical Values Or Text List is a list of attributes associated with all

finite element shapes of the given type on the specified geometric objects. We give examples of these PATRAN commands in the paragraphs below.

2. Neutral File

PATRAN communicates geometry and structure information to other codes through its neutral file. We discuss other PATRAN files for communicating patch area information, generating output results, and replaying an interactive session in the paragraphs below and in Section VIII. For communication with other analysis codes, the application-independent neutral file contains information on geometric entities, such as grids, lines, and patches; finite element nodes and shapes, such as BAR, TRI, and QUAD; and finite element property values. A BAR element connects two finite element nodes; a TRI element connects three nodes; and a QUAD element connects four nodes. Other analysis codes communicate with PATRAN by generating a neutral file.

The neutral file is a strictly formatted text file. The file contains groups of lines describing different graphics entities. For example, groups of lines describing geometric lines or patches or groups specifying finite element nodes or shapes may exist in a typical neutral file. Each group contains a header line with the following integer information: Packet Type, Identification Number, Additional ID, Card Count, and five additional fields for other information. A 0 in any of these fields indicates that the field is not in use. There are numerous Packet Types. We will use the following Packet Types: 1 for a finite element node, 2 for a element shape, 4 for element property specification, 31 for grid data, 32 for line data, and 33 for patch data. Identification Number contains the unique number for a grid, line, patch, node, or finite element shape. Card Count specifies the number of additional cards in the group following the header card. Later we will give examples of entries in the other fields.

3. NEC Input File

The eventual goal of creating a structure in GPK is to produce a NEC formatted input file. At this point, we will give a brief description of the format of a NEC input file [7-3]. The user should refer to NEC's user manual for a full description of NEC's input file format.

NEC's input file contains lines of text describing a structure, followed by additional lines controlling the flow of the analysis code. Each line contains a two-character header, which specifies the function of the line of text; five-character integer fields for integer values; and ten-character real fields for real values.

A surface patch (SP) card is an example of a text line specifying a structure. After the SP characters, the card contains an integer: 0 for an arbitrarily shaped surface patch, 1 for a rectangular flat surface patch, 2 for a triangular patch, or 3 for a quadrilateral patch. Real values for the coordinates of the center of the patch, the elevation and azimuthal angles for the normal vector at the center, and the patch area follow a 0 in the integer field. Real values for the coordinates of corner points follow a 1, 2, or 3 in the integer field. For these three cases an SC card must follow an SP card to provide room for the additional real fields. We illustrate examples of structure cards in Section VII.C.4.

A excitation (EX) card is an example of a program control card. A 1 in the first integer field specifies a plane wave excitation. The number of θ angles and number of ϕ angles follow the first integer field. Real values for the direction and polarization angles follow the integer fields. Other types of excitations are possible. We illustrate examples of program control cards in Section VII.C.4.

4. Creating Patches and Wires for NEC

In the current implementation of EIAW, there are three methods to specify a scattering or radiating structure. These three methods correspond to the three types of NEC elements: arbitrarily shaped surface patches, flat surface patches defined by their corner nodes, and straight wire segments. An arbitrary surface patch is a structure element defined by its area, the coordinates of its center point, and the orientation of a normal vector at its center. The geometric two-dimensional patches in PATRAN will correspond to the arbitrarily shaped surface patches in NEC. The TRI and QUAD finite element shapes in PATRAN will correspond to the flat surface patches in NEC. The BAR finite element shapes and their associated property specifications in PATRAN will correspond to wire segment sets in NEC.

Figure 7.5 illustrates a typical set of PATRAN geometric input commands. The terse commands specify an octant of a sphere using the arbitrarily shaped surface patch method. Commands (1) and (2) instruct the graphics package to create a 90^{o} arc of radius 1 in the x-z plane. Command (3) instructs the graphics package to rotate the arc about the z axis 15^{o}. Commands (4), (6), (8), (10), and (12) break the 15^{o} wedge into ten smaller patches. Figure 7.6 shows the ten patches labeled 10 to 19. Commands (5), (7), (9), (11), and (13) remove the old patches after every break. The last command, (14), replicates the ten patches on the wedge to make a total of sixty patches for the octant of the sphere. Figure 7.6 illustrates the resulting set of sixty patches.

GPK must pass information about the objects in its data base to translators and analysis codes. For arbitrarily shaped surface patches, PATRAN creates an area output file and a neutral file containing grid point and surface patch information. Figure 7.7 shows a truncated copy of a PATRAN area output file. The patch ID appears in the first column and the area of each surface patch appears in the second column. Other columns, which NEC does not require, contain further information on the curvature and twist of each patch. Figure 7.8 contains a truncated copy of a neutral file. The first section contains data groups for grid entities. Each group contains a grid ID and spatial coordinate information. A geometric line is a curved one-dimensional object placed in the three-dimensional modeling space. The second section contains a data set for a line. The set contains a line ID, twelve parametric line coefficients, and endpoint grid IDs. The last section contains a data set for a geometric patch. The set contains a patch ID, forty-eight parametric patch coefficients, and four corner grid IDs. Each of the sixty patches in the sphere octant example produces a similar data set.

NEC has the ability to analyze symmetric structures by reflecting the symmetric elements among planes perpendicular to the x, y, or z axis. We will use the above model of one-eighth of a sphere to construct a perfectly conducting sphere scatterer. In the subsequent discussion of the Electromagnetic Interactive Analysis Workstation (EIAW), we will continually return to this example of the sphere scatterer.

```
      . . .
(1)  GR,#,,1
(2)  LI,#,ARC,3(0)/0/1/0/-90,1
(3)  PA,#,ARC,3(0)/0/0/1/15.0,1
(4)  PA,2#,BR,.3333/.3333/2,1
(5)  PA,1,DEL
(6)  PA,2#,BR,.5/.5/2,3
(7)  PA,3,DEL
(8)  PA,2#,BR,.5/.5/2,2
(9)  PA,2,DEL
(10) PA,4#,BR,.5/.5/2,4T5
(11) PA,4T5,DEL
(12) PA,8#,BR,.5/.5/2,6T9
(13) PA,6T9,DEL
(14) PA,50#,RO,5(0)/1/15.0,10T19
      . . .
```

Figure 7.5. Commands to construct an octant of a unit sphere

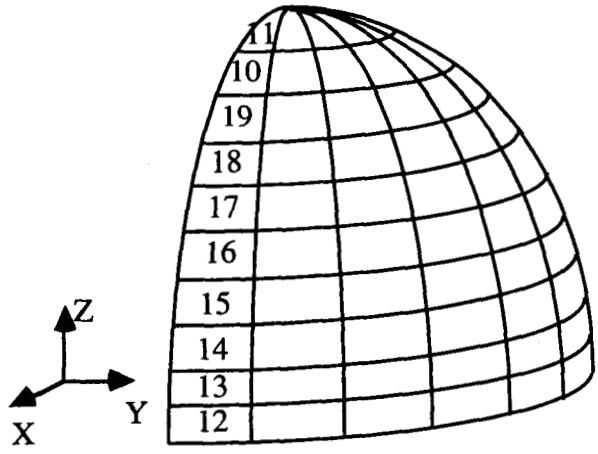


Figure 7.6. Octant of a sphere and the first 10 geometric patches

The next example, an octant of a cube, illustrates the flat surface patch modeling method. Figure 7.9 gives the commands to construct this eighth of a cube. Command (1) instructs the graphics package to create grid point 1 at (0.5,0.0,0.0). Command (2) makes the translated grid point 2 at (0.5,0.5,0.0). Command (3) creates the translated grid points 3 and 4 at (0.5,0.0,0.5) and (0.5,0.5,0.5). Command (4) replicates the existing grid points at locations displaced -0.5 units in the x direction. Figure 7.10 shows the location of these eight grid points. Commands (5) through (7) connect these eight grid points with three geometric patches. The order of the grid points for each patch construction assures

		PATCH CHARACTERISTICS				
PATCH	AREA	MIN ASPECT RATIO	MAX ASPECT RATIO	MIN MESH DISTORTION	MAX MESH DISTORTION	MIN OBLIQUITY
10	0.26379e-01	0.28221e+00	0.49725e+00	0.73886e+00	0.12528e+01	-0.26909e-05
11	0.94195e-02	0.18207e- 01	0.24629e+00	0.14614e+00	0.18225e+01	-0.55371e-05
12	0.35520e-01	0.18995e+01	0.19321e+01	0.98336e+00	0.10162e+01	-0.32068e-06
13	0.34069e-01	0.19188e+01	0.19336e+01	0.97883e+00	0.10207e+01	-0.44947e-06
14	0.32313e-01	0.18619e+01	0.19161e+01	0.97362e+00	0.10258e+01	-0.10674e-05
15	0.30247e-01	0.17632e+01	0.18531e+01	0.96733e+00	0.10320e+01	-0.10185e-05
16	0.27864e-01	0.16271e+01	0.17485e+01	0.95956e+00	0.10397e+01	-0.14235e-05
17	0.25153e-01	0.14588e+01	0.16077e+01	0.94974e+00	0.10494e+01	-0.15064e-05
18	0.22123e-01	0.12623e+01	0.14350e+01	0.93683e+00	0.10622e+01	-0.21806e-05
19	0.18785e-01	0.10417e+01	0.12347e+01	0.91884e+00	0.10801e+01	-0.20683e-05
	. . .					

Figure 7.7. Portion of an output file containing patch area information

that the direction of the patch normal is outward. The GFEG command (8) arranges a set of 4 x 4 finite element nodes on each geometric patch object. The CFEG command (9) connects these finite element nodes with quadrilaterally shaped finite elements. These quadrilateral elements represent the flat surface patches for NEC. Figure 7.10 illustrates the resulting octant of a cube.

This flat patch modeling method is easier than the previous arbitrary patch shape method. This second method usually requires fewer geometric patches and utilizes GPK's finite element generation capability to create multiple NEC patches. The cube object requires only three geometric patches. The previous sphere example requires sixty patches.

This second modeling method does not require an area output file, but it produces distinct entries in the application-independent neutral file. Figure 7.11 illustrates the neutral file for the octant of a cube. The first section contains data groups for finite element nodes. Each group contains a finite element node ID and spatial coordinates. The second section contains data groups for finite element shapes. Each set of three lines contains a finite element ID, a shape type ID in the Additional ID field, a property ID, and the corner nodes associated with each element. The shape type ID is 3 for TRI element shapes or 4 for QUAD element shapes. Each of the twenty-seven patches in the cube octant example produces a data set of three lines.

```
. . .
31      1      0      1      0      0      0      0      0
 0.100000000e+01 0.000000000e+00 0.000000000e+00
31      2      0      1      0      0      0      0      0
 0.437113918e-07-0.437113847e-07 0.100000000e+01
31      3      0      1      0      0      0      0      0
 0.965925813e+00 0.258819074e+00 0.000000000e+00
31      4      0      1      0      0      0      0      0
 0.863496721e+00 0.000000000e+00 0.504674614e+00
31      5      0      1      0      0      0      0      0
 0.834073782e+00 0.223489419e+00 0.504674614e+00
31      6      0      1      0      0      0      0      0
 0.504741371e+00 0.000000000e+00 0.863457561e+00
31      7      0      1      0      0      0      0      0
 0.487542808e+00 0.130636707e+00 0.863457561e+00
31      8      0      1      0      0      0      0      0
 0.964285970e+00 0.000000000e+00 0.265814126e+00
31      9      0      1      0      0      0      0      0
 0.931428731e+00 0.249575600e+00 0.265814126e+00
31     10      0      1      0      0      0      0      0
 0.707133889e+00 0.000000000e+00 0.707079470e+00
. . .

32      1      0      3      0      0      0      0      0
 0.100000000e+01 0.437113918e- 07-0.284114151e-06-0.165685451e+01 0.000000000e+00
-0.437113847e- 07-0.102140518e- 13-0.724234041e-07-0.874227765e- 07 0.100000000e+01
 0.165685439e+01 0.415248337e- 06       1      2
. . .

33     10      0     10      0      0      0      0      0
 0.504741371e+00 0.487542808e+00 0.298023224e- 07-0.342493653e- 01 0.265852332e+00
 0.256793678e+00 0.104308128e- 06-0.180394948e- 01-0.222229704e+00-0.214657486e+00
-0.931322574e- 06 0.150791183e- 01-0.253959417e+00-0.245306015e+00-0.208616257e- 05
 0.172319114e- 01-0.220622134e- 07 0.130636707e+00 0.132330075e+00 0.127821013e+00
-0.252795296e- 07 0.688076541e- 01 0.696995780e- 01 0.673245489e- 01 0.242946285e- 08
-0.575173348e- 01-0.582627691e- 01-0.562776551e- 01-0.984434311e- 08-0.657295882e- 01
-0.665814504e- 01-0.643127113e- 01 0.863457561e+00 0.863457561e+00 0.000000000e+00
 0.000000000e+00 0.964275420e+00 0.964275420e+00 0.000000000e+00 0.000000000e+00
 0.130186439e+00 0.130186439e+00 0.000000000e+00 0.000000000e+00 0.698603391e- 01
 0.698603391e- 01 0.000000000e+00 0.000000000e+00       6     12     13      7
. . .
```

Figure 7.8. Portion of a neutral file containing arbitrarily shaped patch information

```
...
(1) GR,1,,.5
(2) GR,2,TR,0/.5,1
(3) GR,3/4,TR,0/0/.5,1/2
(4) GR,5T8,TR,-.5,1T4
(5) PA,1,QUAD,,1/3/4/2
(6) PA,2,QUAD,,2/4/8/6
(7) PA,3,QUAD,,3/7/8/4
(8) GFEG,1PT3,,4/4
(9) CFEG,1PT3,QUAD/4
...
```

Figure 7.9. Commands to construct an octant of a unit cube

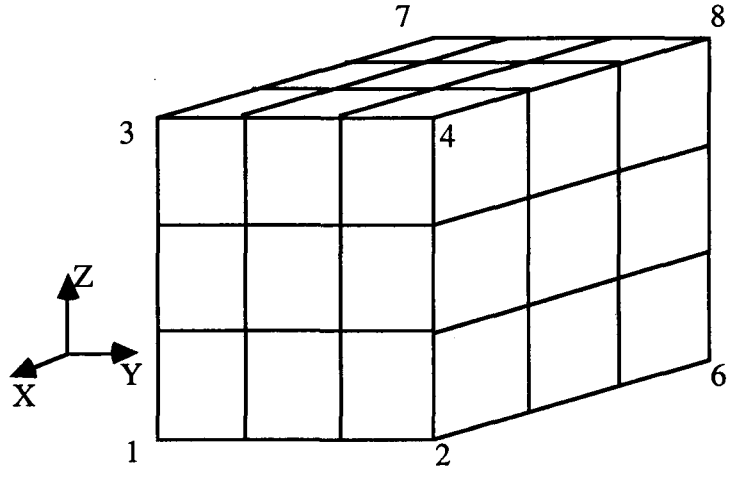


Figure 7.10. Octant of a unit cube with corner grid points

The next example, a set of wires beginning at (0.0,0.0,0.0) and ending at (0.5,0.5,0.5), illustrates the wire modeling feature. Figure 7.12 gives the commands to construct this set of wires. Command (1) instructs the graphics package to create a straight line from the origin to (0.5,0.5,0.5). Command (2) creates five nodes on the line such that the distance between a pair of nodes is twice the distance between the previous pair. 8.0 is the ratio of the distance between the last pair of nodes and the first pair of nodes. Command (3) connects nodes on the line with finite element BAR shapes. Each of these finite element BAR shapes may represent multiple wire segments, depending on the next command. Command (4) assigns a property ID number to all BAR shapes on this geometric line entity. The property list is 2.0/1.0/0.03/0.0/0.0/0.0. We give the meaning of this list below. Figure 7.13 illustrates this set of wires.

```
 1     1     0     2     0     0     0     0     0
 0.500000000e+00 0.000000000e+00 0.000000000e+00
1G     6     0     0  000000
 1     2     0     2     0     0     0     0     0
 0.500000000e+00 0.166666672e+00 0.000000000e+00
1G     6     0     0  000000
 1     3     0     2     0     0     0     0     0
 0.500000000e+00 0.333333343e+00 0.000000000e+00
1G     6     0     0  000000
 1     4     0     2     0     0     0     0     0
 0.500000000e+00 0.500000000e+00 0.000000000e+00
1G     6     0     0  000000
 1     5     0     2     0     0     0     0     0
 0.500000000e+00 0.000000000e+00 0.166666672e+00
1G     6     0     0  000000
 1     6     0     2     0     0     0     0     0
 0.500000000e+00 0.166666672e+00 0.166666672e+00
1G     6     0     0  000000
 1     7     0     2     0     0     0     0     0
 0.500000000e+00 0.333333343e+00 0.166666672e+00
1G     6     0     0  000000
. . .

 2     1     4     2     0     0     0     0     0
     4     0     0     0 0.000000000e+00 0.000000000e+00 0.000000000e+00
     1     2     6     5
 2     2     4     2     0     0     0     0     0
     4     0     0     0 0.000000000e+00 0.000000000e+00 0.000000000e+00
     2     3     7     6
. . .
```

Figure 7.11. Portion of a neutral file containing finite element and node information for the cube octant

```
    . . .
(1) LI,#,,.5/.5/.5/0/0/0
(2) GFEG,1L,G,5/8.0
(3) CFEG,1L,BAR
(4) PFEG,1L,BAR,2.0/1.0/0.03/0.0/0.0/0.0
    . . .
```

Figure 7.12. Commands to create wire elements

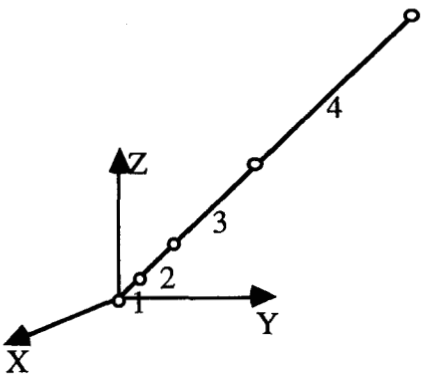


Figure 7.13. Set of four wire segments modeled in GPK

The property list has meaning to the translator and the NEC input file. The first 2.0 is a header for all wire property lists. 1.0 gives the number of wires represented by each finite element BAR shape. If this field were greater than 1.0, each of the four BAR shapes would contain multiple wire segments. 0.03 is the uniform radius of all wires represented by one BAR shape. If this field were 0.0, the field would signal to the translator and NEC that the set has a tapered wire radius. The first 0.0 gives the ratio of the length of the next wire to the length of the previous wire. The second 0.0 gives the radius of the first wire in the set. The third 0.0 gives the radius of the last wire in the set. These last three fields would contain nonzero entries if the set had a tapered wire radius.

At this point we will explain the parameters for a set of wires in a NEC input file entry. A NEC wire entry requires the ratio, R, of the length of the next wire over the length of the previous wire. The property list, which we describe above, gives this number. If the total length of all wire segments is L and R equals 1.0, then equation 7.1 gives the length, S_1, of the first wire segment. N is the number of wires in the set. If R is not 1.0, equation 7.2 gives the length of the first segment.

$$S_1 = \frac{L}{N} \tag{7.1}$$

$$S_1 = \frac{L}{1 + R + \ldots + R^{N-2} + R^{N-1}} = \frac{L\,(1-R)}{1 - R^N} \tag{7.2}$$

Equation 7.3 gives the length, S_i, of the other i^{th} segment in the set of wires.

$$S_i = S_1 R^{i-1} \tag{7.3}$$

Finally, equation 7.4 gives the ratio, T, of the wire radii of the next wire segment over the previous wire segment. R_1 is the radius of the first wire and R_2 is the radius of the last wire.

$$T = \left(\frac{R_2}{R_1}\right)^{1/(N-1)} \tag{7.4}$$

This third modeling method does not require an area output file, but it produces distinct entries in the application-independent neutral file. Figure 7.14 illustrates the neutral file for the set of wire segments. The first section contains groups of three lines for nodes. Each group of three contains a finite element node ID and spatial coordinates. The second section contains data groups for finite element shapes. Each group contains a finite element ID, a shape type ID in the Additional ID field, a property ID, and the endpoint nodes associated with each BAR element. The shape type ID must equal 2 for BAR element shapes. Each of the four finite element BAR shapes in the above example requires a data set of three lines. The last section of the file contains a data group for element property data. This group contains a property ID number and a list of the numbers associated with that property ID.

5. Modifying Existing Structures

A designer may wish to change an existing scattering or radiating structure to modify the calculated radiation patterns. Therefore, the structure editor must have the ability to bring up an existing object for modification.

There are several ways to recall an existing object for modification. The specification of an old structure exists as a PATRAN data base. The user may read this data base during a design session. The neutral file alone contains all the information specifying a structure. The user may choose to recall the structure by this method. During

```
. . .
1    11     0     2     0     0     0     0     0
 0.000000000e+00 0.000000000e+00 0.000000000e+00
1G     6     0     0  000000
 1    12     0     2     0     0     0     0     0
 0.333333343e+00 0.333333343e+00 0.333333343e+00
1G     6     0     0  000000
 1    13     0     2     0     0     0     0     0
 0.100000012e+01 0.100000012e+01 0.100000012e+01
1G     6     0     0  000000
 1    14     0     2     0     0     0     0     0
 0.233333349e+01 0.233333349e+01 0.233333349e+01
1G     6     0     0  000000
 1    15     0     2     0     0     0     0     0
 0.500000000e+01 0.500000000e+01 0.500000000e+01
1G     6     0     0  000000
. . .

 2     1     2     2     0     0     0     0     0
     2     0     1     0 0.000000000e+00 0.000000000e+00 0.000000000e+00
    11    12     0     0
 2     2     2     2     0     0     0     0     0
     2     0     1     0 0.000000000e+00 0.000000000e+00 0.000000000e+00
    12    13     0     0
 2     3     2     2     0     0     0     0     0
     2     0     1     0 0.000000000e+00 0.000000000e+00 0.000000000e+00
    13    14     0     0
 2     4     2     2     0     0     0     0     0
     2     0     1     0 0.000000000e+00 0.000000000e+00 0.000000000e+00
    14    15     0     0
. . .

 4     1     2     2     2     2     0     6     0
0.200000000e+01 0.100000000e+01 0.300000000e-01 0.000000000e+00 0.000000000e+00
0.000000000e+00
. . .
```

Figure 7.14. Portion of a neutral file with information on wire segments

a design session, PATRAN automatically creates a session file of all interactive commands. Although this method is lengthy, the user can replay the particular session that created the old structure.

The use of translators, discussed in the following paragraphs, allows another alternative to recall an existing object. The geometry section of a NEC input file contains all the information for the structure. The user can use the NEC to GPK translator on an

existing NEC input file and produce a application-independent neutral file. The user then reads the neutral file of the structure.

Once PATRAN has the old structure, the user can use the object creation, finite element meshing, and property tagging commands to add to or modify a structure.

D. TRANSLATORS

Each software has its own input and output file format. For different codes to exchange data, the user must format the data in a manner readable by the particular program. We call the programs that perform this task translators. However, there may not be a one to one correspondence between the data that one piece of software puts out and the data another piece of software reads. For these cases, the translator program must perform calculations on the raw data before formatting can occur.

We could not expect GPK to produce output files appropriate for the NEC analysis code. Likewise, GPK has no ability to produce an image given a typical NEC input file. Therefore, we produced a NEC to GPK translator and a GPK to NEC translator.

We invoke the translators by choosing the File Translation option in the Setup Menu of the EIAW. Figure 7.15 shows this menu option for a typical translation session. The particular session creates the geometry section of the NEC input file for the sphere octant.

1. Patch Equations

In order to understand the work of the translators, we first present the mathematical description of a curved surface patch within PATRAN. These equations are especially important for the translation of structures created with the arbitrarily shaped patch model described above.

PATRAN describes a curved surface patch by functions of two parametric variables, ξ_1 and ξ_2. These variables range from 0.0 to 1.0. The coordinates of any point

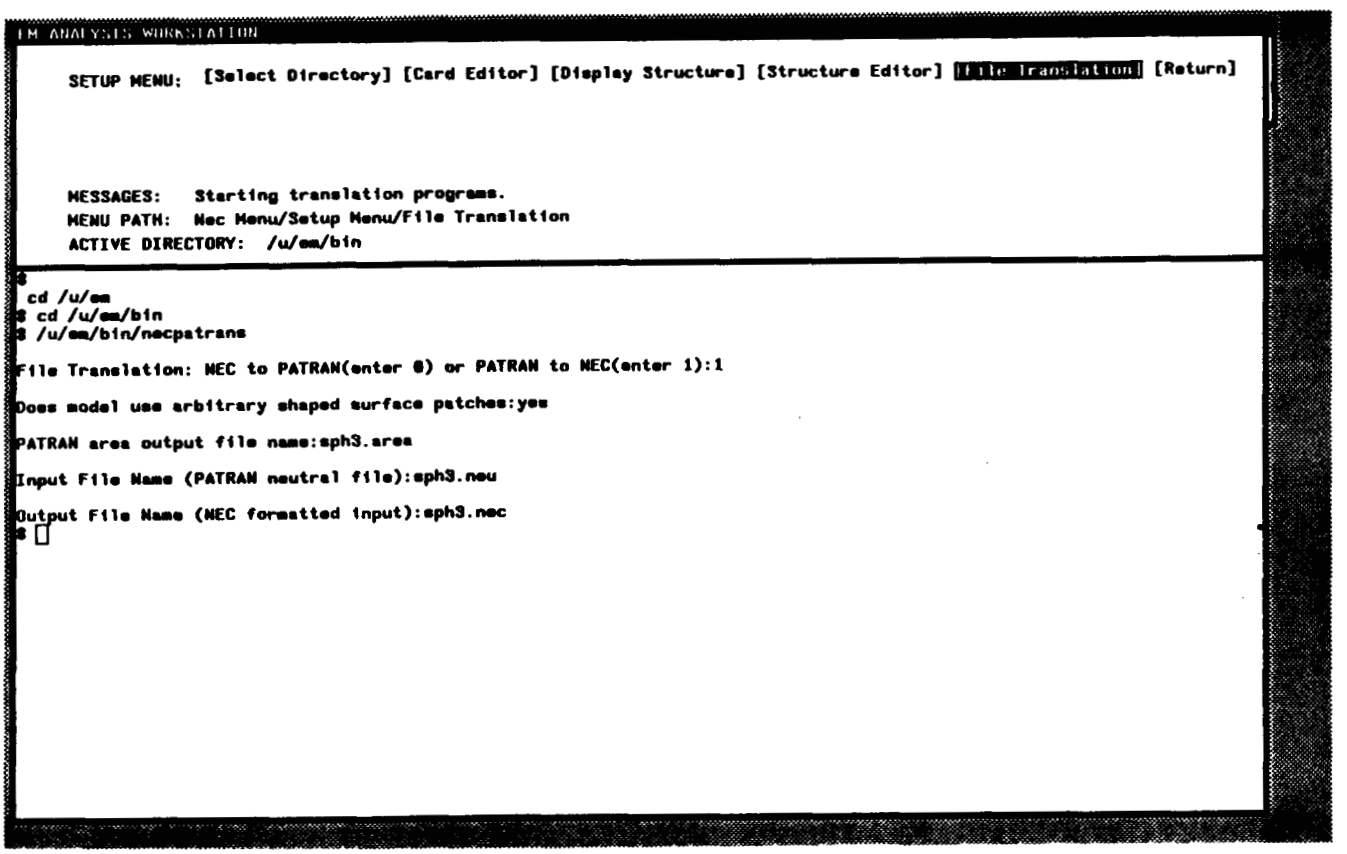


Figure 7.15. Main window of the EIAW showing the Setup Menu and the File Translation option

on the patch are ($x(\xi_1,\xi_2)$, $y(\xi_1,\xi_2)$, $z(\xi_1,\xi_2)$). The coordinates of the corners of the patch are (x(0,0), y(0,0), z(0,0)), (x(1,0), y(1,0), z(1,0)), (x(0,1), y(0,1), z(0,1)), and (x(1,1), y(1,1), z(1,1)).

Knowledge of the values of x, y, and z and the partial derivatives of x, y, and z, with respect to the parametric variables, at the corners of the curved patch determines the values of x, y, and z within the body of the patch [7-1]. Let $Z(\xi_1,\xi_2)$ represent any one of $x(\xi_1,\xi_2)$, $y(\xi_1,\xi_2)$, or $z(\xi_1,\xi_2)$. Equation 7.5 gives the coordinates of points within the body of the patch.

$$Z(\xi_1,\xi_2) = F(\xi_1)\ \beta\ F^T(\xi_2) \tag{7.5}$$

Equations 7.6 and 7.7 give the partial derivatives of the coordinates with respect to parametric variables.

$$Z_{,\xi_1}(\xi_1,\xi_2) = F'(\xi_1)\ \beta\ F^T(\xi_2) \tag{7.6}$$

$$Z_{,\xi_2}(\xi_1,\xi_2) = F(\xi_1)\ \beta\ F^{T'}(\xi_2) \tag{7.7}$$

Equations 7.8 through 7.12 give the expressions for the F functions in terms of a parametric variable.

$$F(\xi) = \left[F_1(\xi)\ F_2(\xi)\ F_3(\xi)\ F_4(\xi)\right] \tag{7.8}$$

$$F_1(\xi) = 2\xi^3 - 3\xi^2 + 1 \tag{7.9}$$

$$F_2(\xi) = -2\xi^3 + 3\xi^2 \tag{7.10}$$

$$F_3(\xi) = \xi^3 - 2\xi^2 + \xi \tag{7.11}$$

$$F_4(\xi) = \xi^3 - \xi^2 \tag{7.12}$$

Equation 7.13 gives the matrix of values of the coordinates and their partials at the corners of the curved surface patch. Such a matrix exists for each of the three Cartesian coordinates. These forty-eight numbers appear as the patch coefficients in the neutral file shown in Figure 7.8.

$$\beta = \begin{bmatrix} Z(0,0) & Z(0,1) & Z_{,\xi_2}(0,0) & Z_{,\xi_2}(0,1) \\ Z(1,0) & Z(1,1) & Z_{,\xi_2}(1,0) & Z_{,\xi_2}(1,1) \\ Z_{,\xi_1}(0,0) & Z_{,\xi_1}(0,1) & Z_{,\xi_1\xi_2}(0,0) & Z_{,\xi_1\xi_2}(0,1) \\ Z_{,\xi_1}(1,0) & Z_{,\xi_1}(1,1) & Z_{,\xi_1\xi_2}(1,0) & Z_{,\xi_1\xi_2}(1,1) \end{bmatrix} \tag{7.13}$$

2. GPK to NEC Translation

The entry in the NEC input file for a arbitrarily shaped surface patch requires the following information: the surface area of the patch, the coordinates of the center of the surface patch, and the elevation and azimuthal angle of the outward normal vector at the center of the patch. We find the surface area of the patch from the output area file (Figure 7.7). We read the forty-eight patch coefficients from the application-independent neutral file (Figure 7.8) to construct the array in equation 7.13 for each coordinate direction. We find the coordinates of the center using equation 7.5 with ξ_1 and ξ_2 set equal to 0.5. To find the elevation angle and azimuthal angle, we use equations 7.6 and 7.7 with ξ_1 and ξ_2 set equal to 0.5. Using equations 7.6 and 7.7, which give the tangent vectors at the center of the patch, we first find the normal vector by taking a cross product and then evaluate the orientation of this normal vector.

Returning to the sphere scatterer example, we can use the GPK to NEC translator to create the geometry section of the NEC input file. Passing the files, portions of which we illustrate in Figures 7.7 and 7.8, for the octant of the sphere scatterer to the translator, we obtain the file illustrated in Figure 7.16. Each line of this portion of the geometry section of the NEC input file contains a specification for an arbitrary shaped surface patch. The nonzero columns correspond to the following information: the Cartesian coordinates of the center of the patch, the elevation and azimuthal angle of the normal vector at the center, and the area of the surface patch.

The entry in the NEC input file for a flat patch requires three corner nodes for a rectangular or triangular patch or four corner nodes for a quadrilateral patch. For a neutral file created using the flat patch model, the translation to a NEC input file is straightforward. The node numbers associated with a finite element TRI or QUAD shape identify the nodes in the neutral file. We then convert the coordinates of the identified nodes to the coordinates of a flat patch for NEC.

```
SP  0   0 0.3859329 0.0508090 0.9214072 67.072242 7.4999895 0.0263790
SP  0   0 0.1345398 0.0177125 0.9908702 82.273910 7.4999633 0.0094195
SP  0   0 0.9891579 0.1302251 0.0684593 3.8678658 7.5000153 0.0355200
SP  0   0 0.9713060 0.1278749 0.2015152 11.563590 7.5000153 0.0340690
SP  0   0 0.9367341 0.1233234 0.3284283 19.165903 7.5000677 0.0323130
SP  0   0 0.8866230 0.1167261 0.4480075 26.654396 7.4999895 0.0302470
SP  0   0 0.8221483 0.1082379 0.5590695 34.039726 7.4999108 0.0278640
SP  0   0 0.7444881 0.0980137 0.6604209 41.353576 7.4998846 0.0251530
SP  0   0 0.6548281 0.0862098 0.7508614 48.641994 7.4999108 0.0221230
SP  0   0 0.5543500 0.0729816 0.8291994 55.955921 7.4999108 0.0187850
. . .
```

Figure 7.16. Portion of an NEC input file for the sphere scatterer

Returning to the cube scatterer example, we can use the GPK to NEC translator to create the geometry section of the NEC input file. Passing the file, portions of which we illustrate in Figure 7.11, for the octant of the cube scatterer to the translator, we obtain the file illustrated in Figure 7.17. Each pair of lines with headers SP and SC contains a specification for a flat surface patch. The nonzero columns correspond to the following information: a 1 specifying a rectangular patch and the Cartesian coordinates of three corner nodes. A 2 in the second column of a SP card would specify a triangular patch followed by the Cartesian coordinates of three corner nodes. A 3 in the second column of a SP card would specify a quadrilateral patch followed by the Cartesian coordinates of four corner nodes. The order of these nodes determines the orientation of a normal and of surface currents.

The entry in the NEC input file for a set of wires requires the number of segments in the set, the beginning and ending points, the wire radius or 0.0 for tapering, the ratio of lengths of adjacent wires, the radius of the first wire, and the radius of the last wire. For a neutral file created using the wire model, the translation to a NEC input file is straightforward. The node numbers associated with a BAR shape identify the nodes in the neutral file. These nodes are the endpoints of wire segments. The property list assigned to the BAR set determines the other wire parameters.

Returning to the wire example, we can use the GPK to NEC translator to create the geometry section of the NEC input file. Passing the file, portions of which we illustrate in Figure 7.14, for the BAR set to the translator, we obtain the file illustrated in Figure 7.18. Each line with header GW contains a specification for a set of wires. The nonzero columns in the GW card correspond to the following information: a 1 specifying the number of

```
SP  0   1 0.5000000 0.000e+00 0.000e+00 0.5000000 0.1666667 0.000e+00
SC  0   0 0.5000000 0.1666667 0.1666667
SP  0   1 0.5000000 0.1666667 0.000e+00 0.5000000 0.3333333 0.000e+00
SC  0   0 0.5000000 0.3333333 0.1666667
SP  0   1 0.5000000 0.3333333 0.000e+00 0.5000000 0.5000000 0.000e+00
SC  0   0 0.5000000 0.5000000 0.1666667
SP  0   1 0.5000000 0.000e+00 0.1666667 0.5000000 0.1666667 0.1666667
SC  0   0 0.5000000 0.1666667 0.3333333
SP  0   1 0.5000000 0.1666667 0.1666667 0.5000000 0.3333333 0.1666667
SC  0   0 0.5000000 0.3333333 0.3333333
SP  0   1 0.5000000 0.3333333 0.1666667 0.5000000 0.5000000 0.1666667
SC  0   0 0.5000000 0.5000000 0.3333333
SP  0   1 0.5000000 0.000e+00 0.3333333 0.5000000 0.1666667 0.3333333
SC  0   0 0.5000000 0.1666667 0.5000000
SP  0   1 0.5000000 0.1666667 0.3333333 0.5000000 0.3333333 0.3333333
SC  0   0 0.5000000 0.3333333 0.5000000
SP  0   1 0.5000000 0.3333333 0.3333333 0.5000000 0.5000000 0.3333333
SC  0   0 0.5000000 0.5000000 0.5000000
. . .
```

Figure 7.17. Portion of an NEC input file for the cube scatterer

```
GW  0   1 0.000e+00 0.000e+00 0.000e+00 0.3333333 0.3333333 0.3333333 0.0300000
GW  0   1 0.3333333 0.3333333 0.3333333 1.0000000 1.0000000 1.0000000 0.0300000
GW  0   1 1.0000000 1.0000000 1.0000000 2.3333333 2.3333333 2.3333333 0.0300000
GW  0   1 2.3333333 2.3333333 2.3333333 5.0000000 5.0000000 5.0000000 0.0300000
. . .
```

Figure 7.18. Portion of an NEC input file for a set of wire segments

segments and the six values for the Cartesian coordinates of the beginning and ending points. 0.03 specifies the radius of each wire in the set. If this value were 0.0, the input will would require a second card, the GC card. The third field of a GC card specifies the ratio of lengths between adjacent wires. The fourth and fifth fields give the radius of the first wire and the radius of the last wire in the set, respectively.

3. NEC to GPK Translation

For GPK to successfully image a structure given in a NEC input file using the arbitrary surface patch model, GPK requires the coordinates of the corner points to create grid entries in its neutral file and the forty-eight patch coefficients which specify the curvature of the patch. However, the only information in the NEC input file is the patch

area and the orientation of the normal vector at the center. There is an infinite number of possible geometric patches that we can construct from this information. Therefore, we make the following assumption about the surface patch geometry. The patch is rectangular with the area specified in the last column of the SP card in the NEC input file. The patch is also tangent at its center to the true surface of the scattering or radiating object.

We mathematically describe the arbitrary patch constructed from the NEC input file with the following equations. Equations 7.14 and 7.15 give the orthogonal unit vectors on the surface of the rectangular patch. α is the elevation angle off the x-y plane. This value is from the sixth numerical entry in the SP card. ϕ is the azimuthal angle off the x axis. This value is from the seventh numerical entry. e_1, e_2, and e_3 are the unit vectors in the direction of the coordinate axes.

$$\hat{n}_1 = \cos(\phi + 90)\,\hat{e}_1 + \sin(\phi + 90)\,\hat{e}_2 \tag{7.14}$$

$$\hat{n}_2 = \cos\phi\cos(\alpha + 90)\,\hat{e}_1 + \sin\phi\cos(\alpha + 90)\,\hat{e}_2 + \sin(\alpha + 90)\,\hat{e}_3 \tag{7.15}$$

Equation 7.16 gives the coordinates of any point on the surface patch as a function of the parametric variables ξ_1 and ξ_2 and the unit vectors given in equations 7.14 and 7.15. The range of these parametric variables are 0.0 to 1.0. Equations 7.17 through 7.19 give the partials of the coordinates with respect to the parametric variables. In the last equation, 7.20, A is the area of the patch and 2w is the length of one side. The area is in the eighth numerical entry in the SP card. The vector, c, is the coordinate of the center of the patch. These coordinates are in the third through the fifth numerical entry in the SP card.

$$\vec{Z}(\xi_1,\xi_2) = (2w\,\xi_1 - w)\,\hat{n}_1 + (2w\,\xi_2 - w)\,\hat{n}_2 + \vec{c} \tag{7.16}$$

$$\vec{Z}_{,\xi_1}(\xi_1,\xi_2) = 2w\,\hat{n}_1 \tag{7.17}$$

$$\vec{Z}_{,\xi_2}(\xi_1,\xi_2) = 2w\,\hat{n}_2 \tag{7.18}$$

$$\vec{Z}_{,\xi_1\xi_2}(\xi_1,\xi_2) = (0,0,0) \tag{7.19}$$

$$w = \frac{\sqrt{A}}{2} \tag{7.20}$$

After calculating the appropriate parameters for the rectangular patch, the translator constructs the neutral file. Using the NEC input file, portions of which we show in Figure 7.16, for the octant of a sphere, the translator produces the neutral file shown in Figure 7.19. The second section contains the forty-eight patch coefficients found using equations

```
31      1      0      1
 4.706851840e-01-1.994197071e-02 8.897709846e-01
31      2      0      1
 4.494856298e-01 1.410846710e-01 8.897709846e-01
31      3      0      1
 3.011806011e-01 1.215599701e-01 9.530434608e-01
31      4      0      1
 3.223801553e-01-3.946667165e-02 9.530434608e-01
. . .

33      1      0     10
 4.706851840e- 01 4.494856298e- 01-2.119955420e- 02-2.119955420e- 02 3.223801553e- 01
 3.011806011e- 01-2.119955420e- 02-2.119955420e- 02-1.483050287e- 01-1.483050287e- 01
 0.000000000e+00 0.000000000e+00-1.483050287e- 01-1.483050287e- 01 0.000000000e+00
 0.000000000e+00-1.994197071e- 02 1.410846710e- 01 1.610266417e- 01 1.610266417e- 01
-3.946667165e- 02 1.215599701e- 01 1.610266417e- 01 1.610266417e- 01-1.952470094e- 02
-1.952470094e- 02 0.000000000e+00 0.000000000e+00-1.952470094e- 02-1.952470094e- 02
 0.000000000e+00 0.000000000e+00 8.897709846e- 01 8.897709846e- 01 0.000000000e+00
 0.000000000e+00 9.530434608e- 01 9.530434608e- 01 0.000000000e+00 0.000000000e+00
 6.327247620e- 02 6.327247620e- 02 0.000000000e+00 0.000000000e+00 6.327247620e- 02
 6.327247620e- 02 0.000000000e+00 0.000000000e+00      1      2      3      4
. . .
```

Figure 7.19. Portion of a neutral file produced by passing an NEC input file through the NEC to GPK translator

7.16 through 7.20. We already discussed in Sections VII.C.2 and VII.C.4 the different sections of the neutral file for an arbitrary patch shape.

The translation of a NEC input file containing flat patch information to a neutral file requires little calculation. The translation process takes one SP and SC card pair and creates one data group for a finite element TRI or QUAD shape and three or four data groups for finite element nodes. Look to Section VII.C.2 for a discussion of the data groups in an application-independent neutral file. Look to Section VII.D.2 for the correspondence between a flat surface patch in NEC and a TRI or QUAD finite element shape in PATRAN.

The translation of a NEC input file containing wire segment information to a neutral file also requires little calculation. The translation process takes one GW and GC card pair and creates one data group for a finite element BAR shape, two data groups for finite element nodes, and one data group for a property specification. Look to Section VII.C.2 for a discussion of the data groups in an application-independent neutral file.

Look to Section VII.D.2 for the correspondence between a set of wire segments in NEC and a BAR finite element shape in PATRAN.

E. CARD EDITOR

The structure editor, together with the GPK to NEC translator, can only create the geometry section of a NEC input file. A complete NEC input requires other cards. Scaling, frequency, and excitation cards are examples. In order to ease the addition of these cards as well as the modification of existing input files, we create a menu and prompt driven interactive file editor, called a card editor.

We want the card editor to run on several different machines with many different applications. To this end, we locate the application-dependent information in a FIELDS file. This file contains formatting information for the different input cards in a particular application. We also locate the terminal-dependent information in an ESCAPES file. The ESCAPES file contains the commands to manipulate the cursor for a particular terminal. The following paragraphs discuss the FIELDS file created for NEC and the ESCAPES file created for Tektronix VT100 and Sun window terminals.

1. FIELDS File for NEC

We can best explain the format and contents of a FIELDS file using the file for NEC as an example. We illustrate a portion of this file in Figure 7.20. The original file contains groups of entries for all the possible NEC input cards. Figure 7.20 illustrates an entry for a SP card and a SC card. The numbers in parenthesis on the left do not appear in the original file. They exist for the purposes of discussing each line entry.

The entry for the SP card begins on line (1) and contains the following information. Line number (1) contains a text string which comments the entry. Line (1) comments that the SP card is the eleventh card specified in this file. Line (2) is a text string identifying a group of cards. The SP card belongs to the geometry group. Line (3) is the number of possible values that could trigger the automatic insertion of another card after the SP card. Lines (4) through (6) are a list of the possible triggers. These lines indicate that if the entry in the third column of an SP card is 1, 2, or 3, the editor will automatically add an SC card

```
. . .
(1)  (11) Card generates a surface patch:
(2)  geometry
(3)  3
(4)  3   1   SC
(5)  3   2   SC
(6)  3   3   SC
(7)  Surface Patch  (SP)
(8)  SP
(9)  9
(10) 1   2   7
(11) To input parameters of a single surface patch.
(12)
(13) 3   5   1
(14) A zero should occur in this field.
(15)
(16) 6   10   1
(17) Select patch shape: 0=arbitrary patch shape 1=rectangle patch shape 2=triangular
(18) patch 3=quadrilateral patch
(19) 11   20   5
(20) Patch shape 0: x coordinate of patch center. Patch shape 1-3: x coordinate of
(21) corner 1.
(22) 21   30   5
(23) Patch shape 0: y coordinate of patch center. Patch shape 1-3: y coordinate of
(24) corner 1.
(25) 31   40   5
(26) Patch shape 0: z coordinate of patch center. Patch shape 1-3: z coordinate of
(27) corner 1.
(28) 41   50   5
(29) Patch shape 0: elevation angle above the x-y plane of normal vector in degrees.
(30) Patch shape 1-3: x coordinate of corner 2.
(31) 51   60   5
(32) Patch shape 0: azimuth angle from the x axis of normal vector in degrees.
(33) Patch shape 1-3: y coordinate of corner 2.
(34) 61   70   5
(35) Patch shape 0: patch area in square of units used.  Patch shape 1-3: z
(36) coordinate of corner 2.
(37) (12) Card generates a surface patch (continue):
(38) geometry
(39) -1
(40) Surface Patch  (SC)
(41) SC
(42) 9
(43) 1   2   7
(44) If the third field in the SP card is 1 or 2 or 3, use this card for additional
(45) parameters.
(46) 3   5   1
(47) A zero should occur in this field.
(48)
(49) 6   10   1
(50) A zero should occur in this field.
(51)
(52) 11   20   5
(53) Patch shape 1-3: x coordinate of corner 3.
(54) Multiple patch: x coordinate of corner 3.
(55) 21   30   5
(56) Patch shape 1-3: y coordinate of corner 3.
(57) Multiple patch: y coordinate of corner 3.
(58) 31   40   5
(59) Patch shape 1-3: z coordinate of corner 3.
(60) Multiple patch: z coordinate of corner 3.
(61) 41   50   5
(62) Patch shape 3 only: x coordinate of corner 4.
(63)
(64) 51   60   5
(65) Patch shape 3 only: y coordinate of corner 4.
(66)
(67) 61   70   5
(68) Patch shape 3 only: z coordinate of corner 4.
     . . .
```

Figure 7.20. Portion of the FIELDS file for the Numerical Electromagnetics Code

after the SP card. Line (7) is the text string used for the menu entry. Line (8) is the first two characters in the SP card. Line (9) is the number of numerical fields in the SP card. We group the field information into sets of three lines. Lines (10) through (34) in steps of 3 each contain three numbers. The first number is the beginning column for the field. The second number is the ending column for the field. There are eighty columns in a NEC card entry. The third number indicates the type of information in the field. 1 indicates integer. 2 indicates normal floating point format. 3 indicates FORTRAN logical format. 4 indicates character format. 5 indicates exponential real format. 6 indicates FORTRAN double precision format. 7 indicates that the user cannot modify the field. Lines between (10) and (37) which have text or blanks contain text displayed to the user during a interactive session.

The entry for the SC card begins on line (37). Like the SP card entry, the SC card entry has similar lines. Line (37) comments that the SC card is the twelfth card specified in this file. Line (38) indicates that the SC card is part of the geometry card group. Line (39) contains a negative number, which indicates that the appearance of an SC card depends on previous cards in the input file. Line (40) shows the menu entry for the SC card. Line (41) shows the two characters at the beginning of an SC card.

2. ESCAPES File for Tektronix VT100 and Sun Window Terminals

We can best explain the format and contents of the terminal-dependent ESCAPES file using the file for a Tektronix VT100 and a Sun window terminal. We illustrate a portion of this file in Figure 7.21. There are groups of entries for the possible escape sequences on a VT100 as well as user-defined character sequences. An escape sequence is a set of keyboard punches containing an escape character. Moving the cursor on the terminal, inserting and deleting text lines, and blanking portions of the text on the screen are examples of the effects of these escape sequences. Toggling the menu of card options on or off, exiting the file, and replacing the normal escape sequences with a shorter set are examples of the effects of user-defined escape sequences. The numbers in parentheses on the left do not appear in the original file. They exist for the purposes of discussing each line entry.

```
(1)  45
(2)  0)Insert.n.spaces.from.cursor:
(3)  ^[#@
(4)  1
(5)  1)Cursor.up:
(6)  ^[#A
(7)  1
(8)  2)Cursor.down:
(9)  ^[#B
(10) 1
(11) 3)Cursor.forward:
(12) ^[#C
(13) 1
(14) 4)Cursor.backward:
(15) ^[#D
(16) 1
(17) 5)Cursor.nth.line.down:
(18) ^[#E
(19) 1
(20) 6)Cursor.location:
(21) ^[#;#H
(22) 1;1
(23) 7)Erase.from.cursor.to.bottom:
(24) ^[J
(25) 8)Erase.from.cursor.to.end.of.line:
(26) ^[K
(27) 9)Insert.n.lines.from.current.line:
(28) ^[#L
(29) 1
(30) 10)Delete.n.lines.from.current.line:
(31) ^[#M
(32) 1
(33) 11)Delete.n.characters.from.cursor:
(34) ^[#P
(35) 1
(36) 12)Select.reverse.image.rendition:
(37) ^[#m
(38) 0
     . . .
(39) 37)Within.the.current.field.move.one.character.to.the.right:
(40) ^r
(41) 38)Within.the.current.field.move.one.character.to.the.left:
(42) ^l
(43) 39)Quit.from.card.editor.without.saving:
(44) ^q
(45) 40)Exit.from.card.editor.saving.changes:
(46) ^e
(47) 41)Save.changes.in.the.current.file.without.exiting:
(48) ^s
(49) 42)Add.a.new.card.above.the.current.cursor.position:
(50) ^n
(51) 43)Delete.a.card.at.current.cursor.position:
(52) ^d
(53) 44)Toggle.menu.of.card.items:
(54) ^m
```

Figure 7.21. Portion of the ESCAPES file for Tektronix VT100 and Sun terminals

The entries in the ESCAPES file contain the following information. Line (1) is the number of terminal- and user-defined sequences. If a user adds more sequences to the list, he must modify the card editor code so that the editor understands the effect of the new sequences. Groups of two or three entries follow line (1). The first line in any group is a comment line describing the function of the character sequence. Lines (2), (5), and (8) are examples of comment lines. The escape or user-defined sequence follows the comment line. A ^[indicates an escape character. However, a ^ is a normal carat character and usually signals a user-defined character sequence. A # character indicates an arbitrary number entry. A % character indicates an arbitrary character string. If either the # or % character appears in the sequence, the third line in the set indicates the default numeric or text values for these arbitrary entities.

3. Window Interface to the Card Editor

Although the user-defined sequences are short, we prefer to use the menu mechanism to run the card editor. A user normally invokes the card editor by choosing the Card Editor option in the Setup Menu of the EIAW. When the card editor program begins, a new window, which contains two smaller subwindows, appears. Look to Section VII.B for a discussion of the window environment on the Sun 3/160 workstation. The top subwindow contains the text of the NEC input file, the menu of card options, and prompts for the contents of each field in a given card. This subwindow is a normal Sun window terminal. The bottom subwindow controls the activity in the top subwindow. This subwindow sends the appropriate character sequences to the top subwindow after a menu selection. Figure 7.22 shows the card editor window of the EIAW.

The card editor has other features. It always highlights the current field in the file. The editor continually updates the display on the bottom of the screen with text explaining the significance of the current field.

Many of the menu options are self explanatory. The FILE menu contains entries to toggle the menu, delete a card entry, or gracefully leave an editing session. The HORI. MOVEMENT and VERT. MOVEMENT control the motion of the cursor from one field to the next. The HORI. MOVEMENT COUNT and VERT. MOVEMENT COUNT control the number of multiple horizontal and vertical cursor movements. The maximum count is thirty. FIELD MOVEMENT and FIELD MOVEMENT COUNT control the cursor

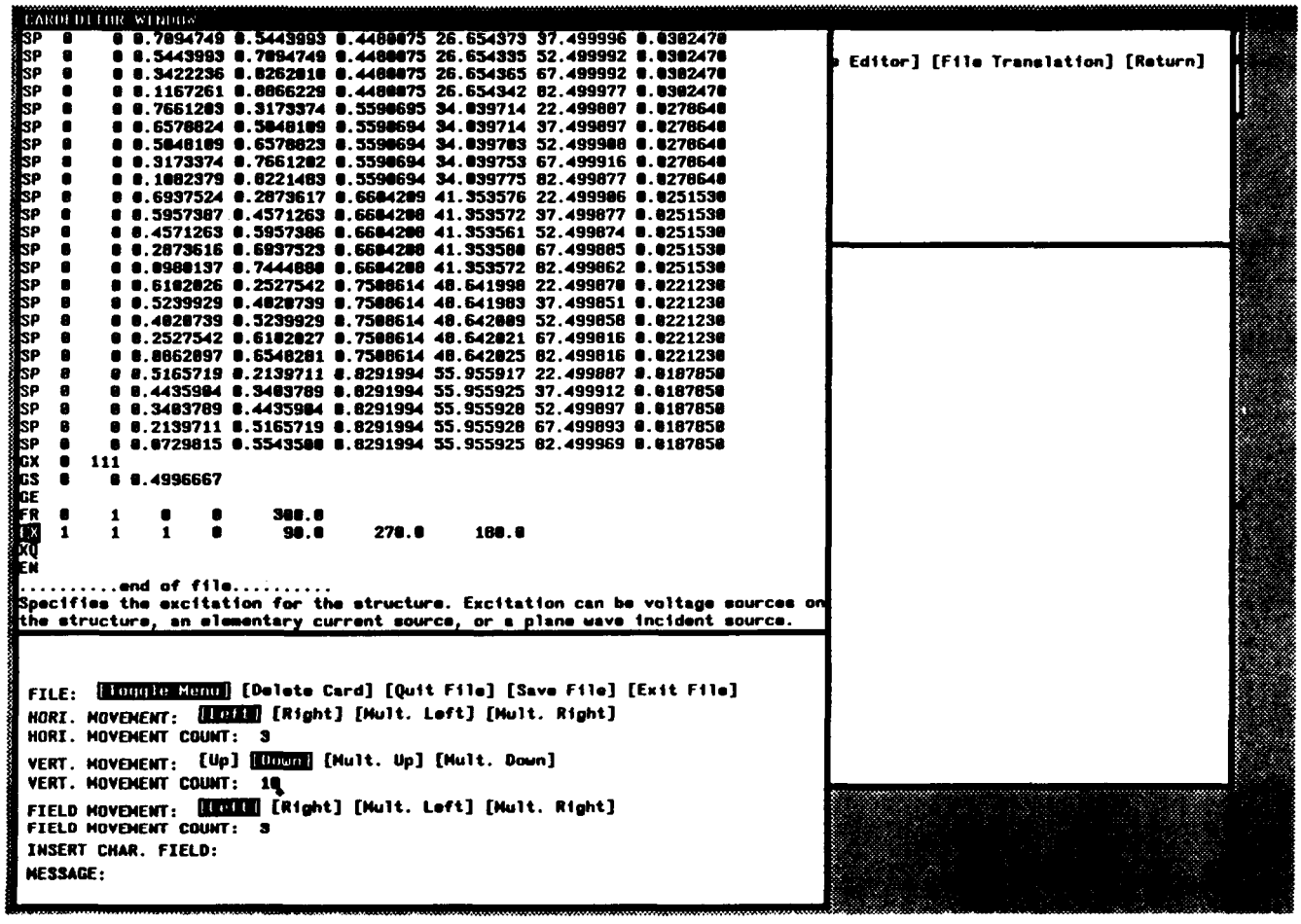


Figure 7.22. The card editor window in the EIAW

movement within a particular field. The user can type in the region after INSERT CHAR. FIELD. The typed characters automatically appear at the current cursor position on the top subwindow.

If the user chooses to toggle the menu, the top subwindow superimposes the menu of card options on top of the current file. The bottom subwindow changes the menus. Figure 7.23 illustrates the appearance of the card menu. The user prompt on the bottom of the terminal subwindow explains the use of the currently selected card. Figure 7.23 illustrates the explanation of the excitation card. The VERT. MOVEMENT and VERT. MOVEMENT COUNT pertain to vertical movement in the card menu.

F. RUN CONTROL

With the input file prepared for NEC, we can run the NEC code on the hypercube. A normal hypercube session requires the user to remotely log into the control processor (CP). On the CP the user enters a queue to await his turn to submit an interactive program. However, instead of remotely logging into another machine, the user of the EIAW would prefer to remain in the EIAW environment. Fortunately, the user can also choose to submit a batch job on the hypercube that would execute at some future time. We will use this batch mechanism to submit hypercube jobs for immediate execution. The only restriction on batch submission is execution delays until the interactive queue is empty. The run control program is the piece of code that causes NEC to execute on the CP and hypercube.

For the run control program to function, it needs information on the location of executable files. It requires an input file. Figure 7.24 illustrates an example input file. Each pair of entries begins with a comment line. The second line of the pair contains the appropriate parameter. This file specifies the control processor machine, the host Sun computer, the parallel and sequential NEC codes, and the location of these codes in the directory hierarchies.

After performing the necessary setup steps, the user can return to the Main Menu and choose the next analysis option, Run Menu. The Run Menu allows the user to remain in the EIAW environment while the NEC code executes on the CP and on the Mark III hypercube.

ORIGINAL PAGE IS
OF POOR QUALITY

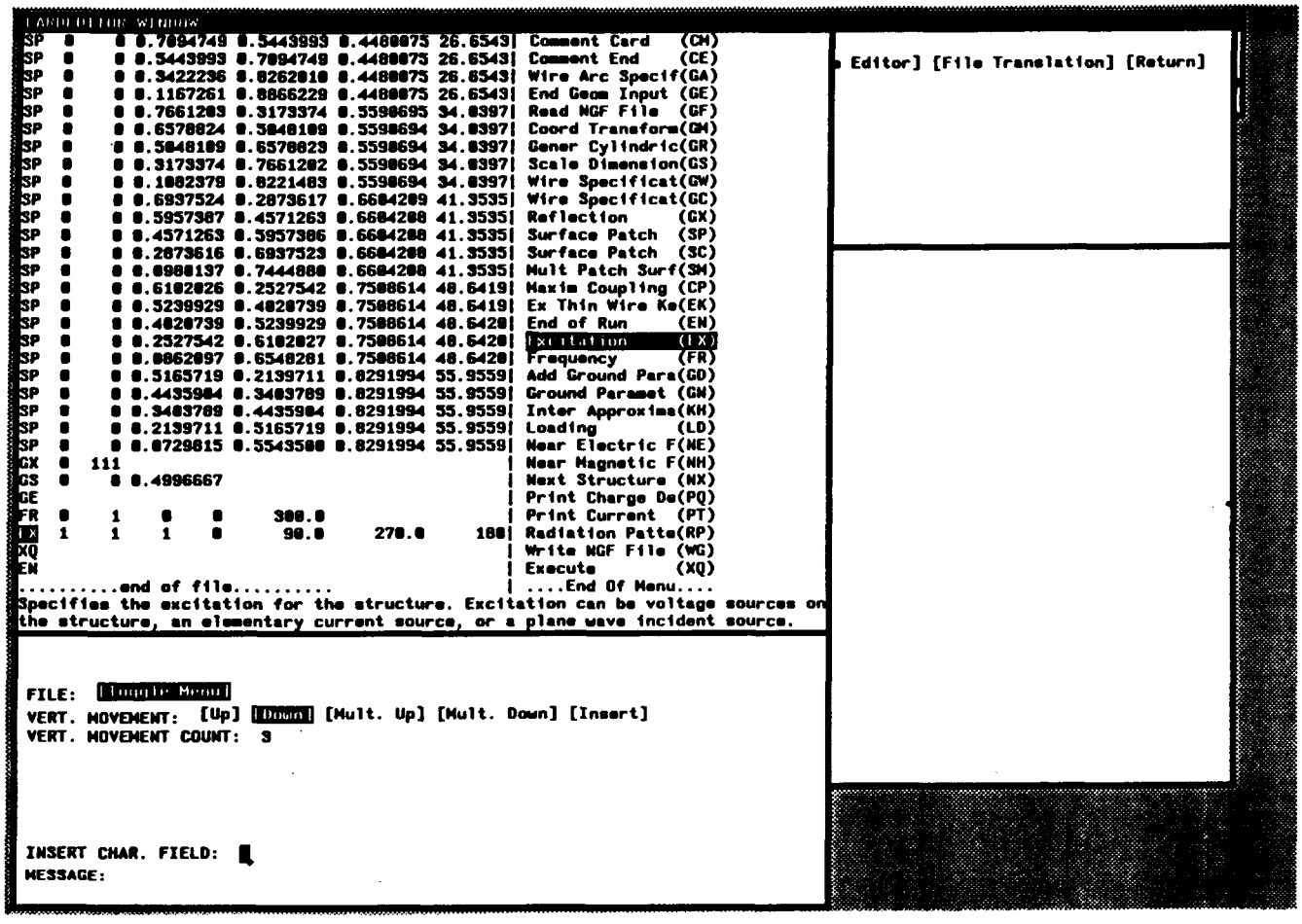


Figure 7.23. The card editor window showing the appearance of the card menu

```
The directory on the Counterpoint containing sequential NEC:
/u/em/jseq
The directory on the Counterpoint containing parallel NEC:
/utmp/emjpar
The name of the NEC sequential code:
neccp.out
The name of the NEC parallel code:
neccp.out
The name of the NEC element code:
neceltp.out
Maximum number of minutes for a batch job on the hypercube:
45
The name of the Counterpoint machine:
cpc5
The name of the Sun machine:
sun9
```

Figure 7.24. Example input file for the run control program

Figure 7.25 illustrates the Run Menu and a typical interactive session with the run control program. During the session the user specifies the NEC input file, sph3.inp; the NEC output file, sph3.out; the file containing the current vector, sph3.cur; the mode of the run, parallel or sequential; and, if parallel mode, the dimension of the hypercube. The run program tells the user that it successfully copied the input file and command file to the CP of the hypercube. The parallel execution of NEC is the current status of the run. When NEC terminates, the run control program will move the solution vector file to the EIAW environment and delete all scratch files from the CP.

G. RESULTS ANALYSIS

With the solution vector, the user can obtain the near electric and magnetic fields, the E θ and E ϕ field, and the total gain. A results generation program takes the solution vector and constructs the appropriate files for GPK to image these fields. The results program will create a neutral file containing the grid on which GPK superimposes the field values at the node points. It also creates a results file which contains the columns of field values. We describe the results generation program in more detail in Section VIII.

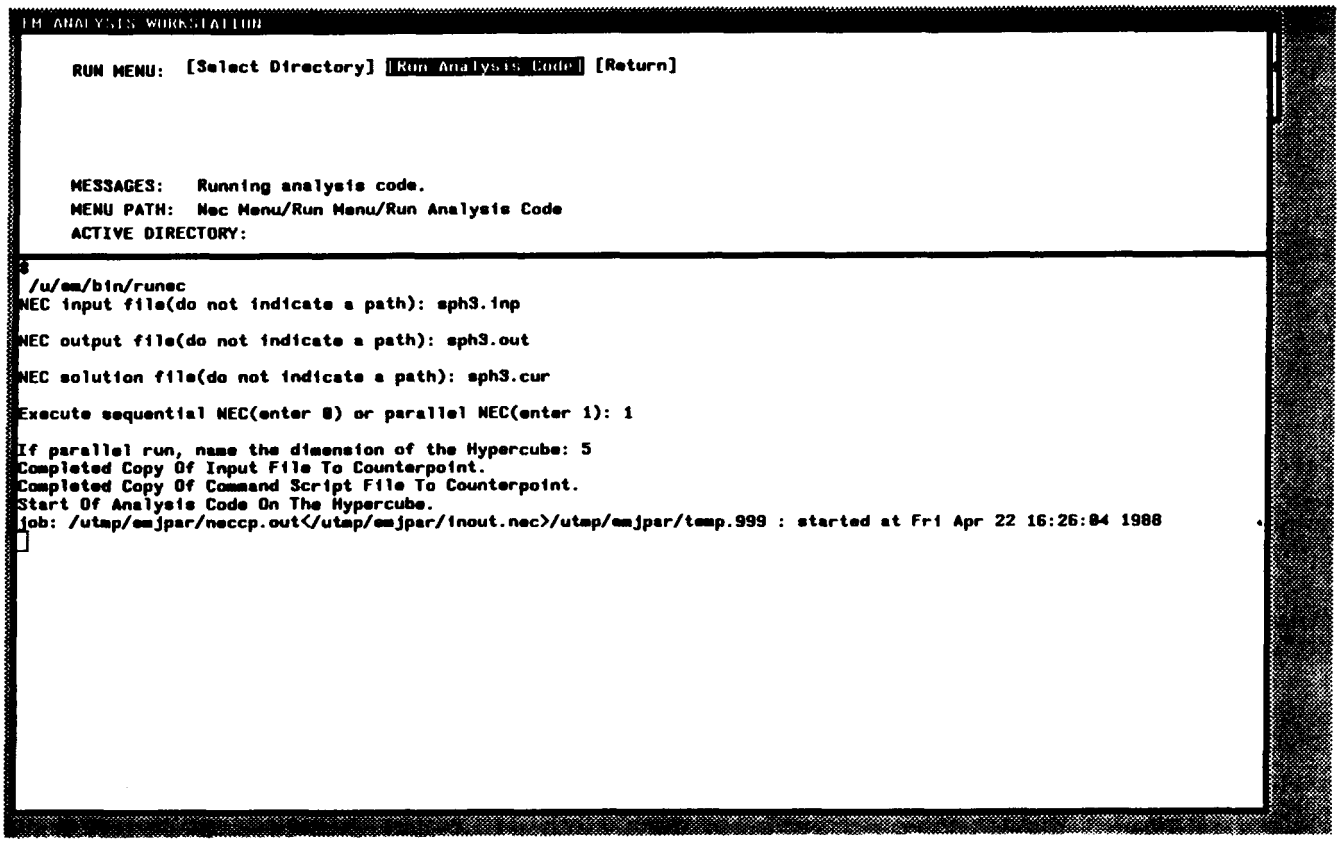


Figure 7.25. The Run Menu and a typical interactive session with the run control program

After recovering the solution vector, the user can return to the Main Menu and choose the next analysis option, Output Menu. The Output Menu allows the user to create files containing field data and display the fields graphically. Figure 7.26 illustrates the Output Menu and a portion of a typical run with the results generator.

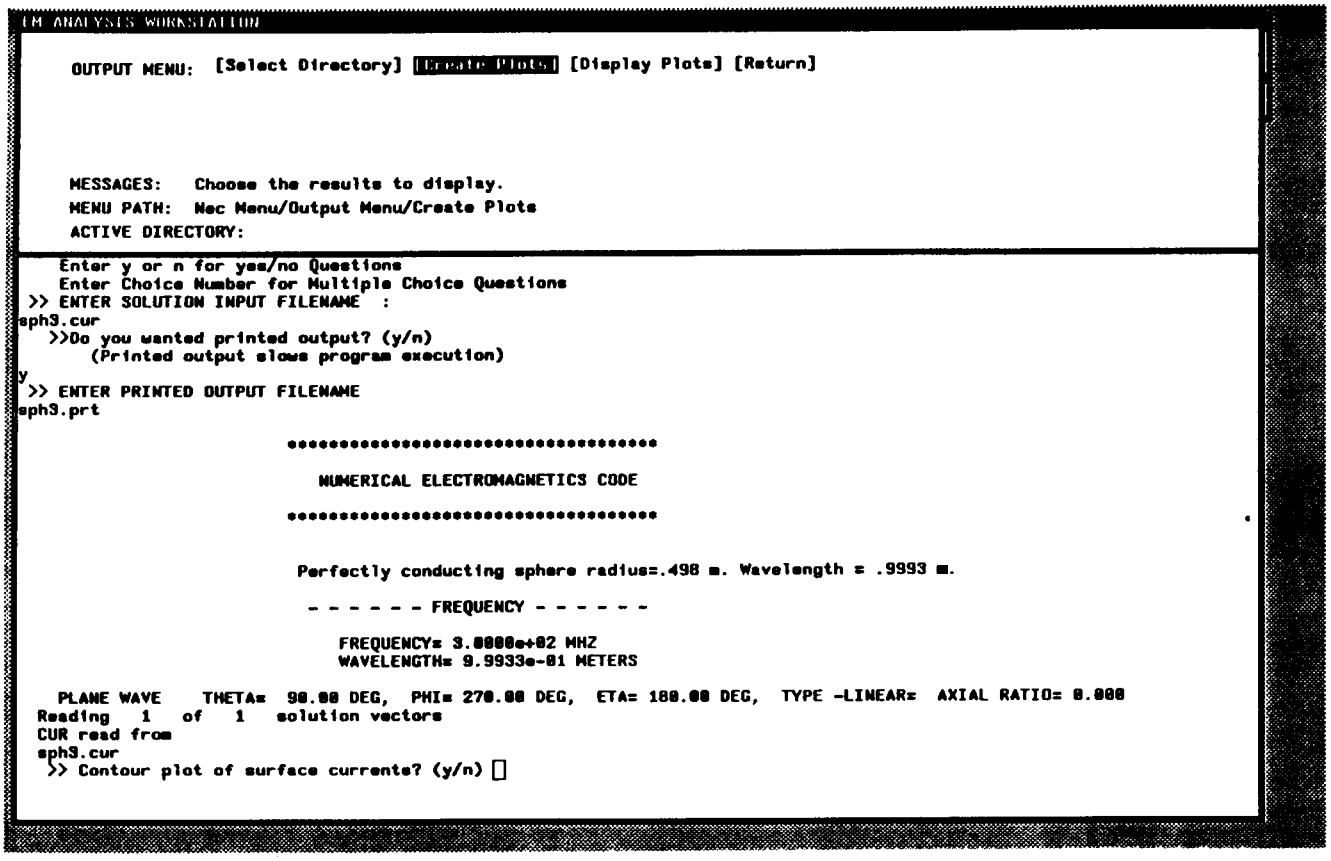


Figure 7.26. The Output Menu and a portion of a typical run with the output generator

REFERENCES

[7-1] SunView Programmer's Guide, Sun Microsystems Inc., Mountain View, CA, October 1986.

[7-2] PATRAN Plus User Manual, Vols. 1 and 2, PDA Engineering, Costa Mesa, CA, July 1987.

[7-3] G. J. Burke and A. J. Poggio, Numerical Electromagnetics Code (NEC) - Method of Moments, Lawrence Livermore National Laboratory, Livermore, CA, 1981.

SECTION VIII

ANALYSIS OF THE NEC RESULTS USING THE ELECTROMAGNETIC INTERACTIVE ANALYSIS WORKSTATION

A. INTRODUCTION

The NEC code, described in Section IV, produces only a printed output file containing information about the currents induced in the object and about the near and far field radiation patterns. A new interactive code, NECPAT, has been developed, running on the Sun workstation computer, which displays results graphically using the solutions computed by the hypercube NEC code. Two distinct types of graphical display are possible. Using NECPAT, simple 1-d line plots of the variation of a specified quantity with one coordinate can be drawn interactively on the workstation screen. NECPAT can also create files containing information for 2-d color contour plots of the variation of a quantity with two coordinates. The graphics package (GPK) uses these files to create color contour plots where color is used to represent the value of the quantity on a 2-d grid. Some knowledge of PATRAN is needed to view the color contour plots. The more experienced PATRAN user will be able to use PATRAN's flexibility and sophistication to make on-screen comparisons of various results from the same or different NEC runs using the files created by NECPAT.

NECPAT's structure is flexible and modular so that additional output options can easily be incorporated and so that NEC and NECPAT can be merged together in the future when it becomes possible to use the Sun computer as a host for the hypercube. In this section, the NECPAT code is described and the available analysis options presented.

B. DESCRIPTION OF THE EIAW CODE NECPAT

The function of the NECPAT code is to provide interactive analysis of NEC results which were previously computed in the hypercube by the NEC code. NECPAT is a sequential code which runs on the EIAW computer. This code, along with the GPK, allows the user to calculate, print, and plot most of the near and far field radiation pattern information which is available as printed output from the original NEC code [8-1]. The

NECPAT code is launched from the EIAW when the user moves to the Output Menu from the Main Menu (Figure 7.3) and from there chooses the Create Plots option.

NECPAT does not solve electromagnetic problems but uses as input solution files computed by the hypercube NEC code. The NEC code itself, described in Section IV and Ref. [8-1], solves a matrix equation to determine the currents induced in a metallic object when the object is excited either by an applied voltage or an incident electromagnetic wave. Specifically, the induced current is expanded in known basis functions, and NEC solves the matrix equation A F = E to determine the amplitudes, F, of these basis functions; here E is the (specified) excitation vector and A is the interaction matrix of order N + 2M where N is the number of wires and M is the number of patches used to model the object. It is this solution vector F which is utilized by NECPAT to provide interactive analysis of the NEC results. The parallel NEC code has been modified so that it now creates a file for NECPAT which contains vector(s) F. This solution file contains all the geometry and excitation cards from the NEC input file used for the run.

When the parallel NEC code is launched from the EIAW, the solution file is automatically passed back across the Ethernet network to the Workstation for use as input to NECPAT. NECPAT calculates the induced currents from the solution vector F and geometry information in the solution file. Using the induced currents, NECPAT interactively calculates the user-requested output.

Figure 8.1 shows a simplified block diagram of the structure of NECPAT in comparison with the structure of the hypercube NEC code. The input portions of the two codes, the portion where the geometry and excitation cards are read and interpreted, are identical. After completing the input portion of the code, the two codes differ. NECPAT reads the solution vector F from the solution file at the point where the hypercube NEC code starts up the element code to fill and solve the matrix equation for F. Following this, the NECPAT calculation of the induced currents from the basis function solution vector is identical to that in the NEC code. NECPAT then queries the user as to what output to create.

An alternative approach to the analysis of NEC results is to specify the desired output before the parallel NEC code is launched and compute this output in the hypercube. This would entail storing the requested current and radiation pattern information in a file.

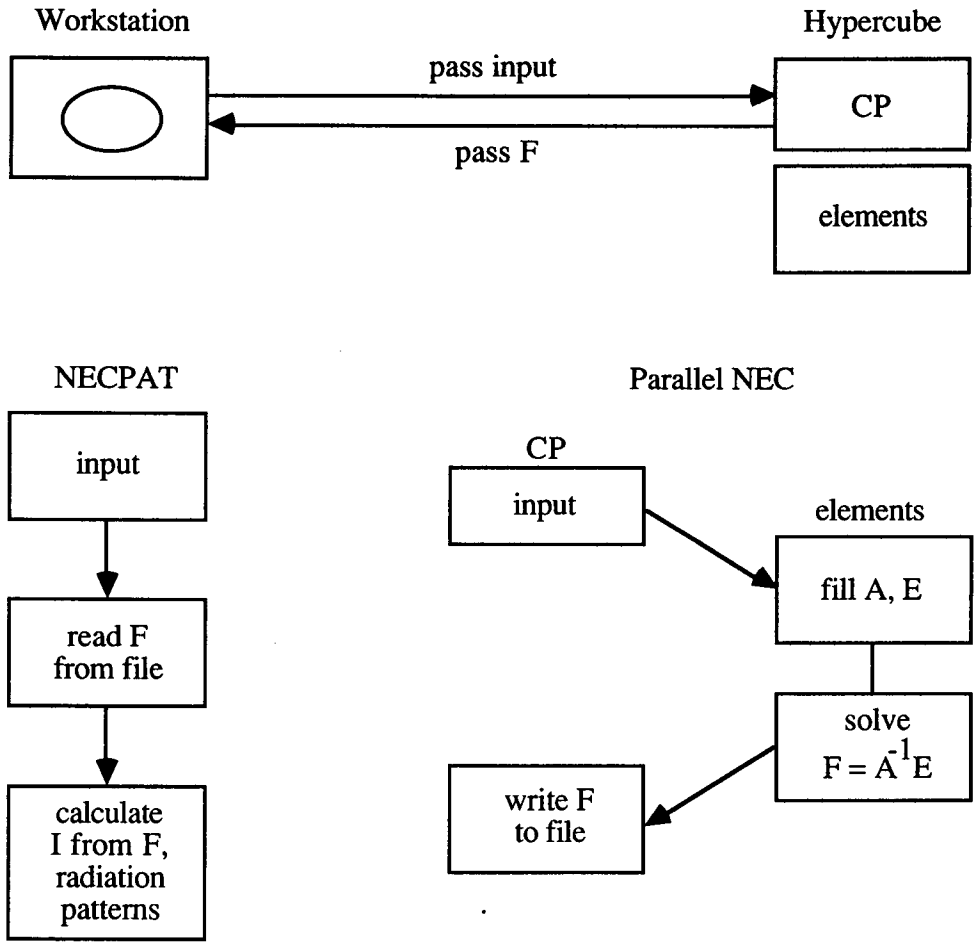


Figure 8.1. Block diagram of the structure of NECPAT in comparison with the structure of the hypercube NEC code

This file would then be passed to the EIAW, where the information would be read and plotted. This approach was not selected for two reasons: 1) The file containing the radiation pattern information would be typically large (1 kilobyte to 1 megabyte per solution), making long-term file storage a problem. 2) Since the output would be specified before the run was made, the user would not have the flexibility of requesting further output, e.g., the same plot with finer resolution, based on the plot he has just seen.

Thus the main advantages of our approach to the analysis of NEC results are that 1) only relatively small solutions files need to be saved, and 2) the user can request additional

output plots at any time. If, on the basis of viewing one plot, the user decides that more resolution or further information is needed, another plot can easily be generated. For comparison, a plot from a previous run also can be generated.

The one disadvantage of our approach is that the user must wait while the Workstation computer calculates the necessary field information from the induced currents.

C. NECPAT OUTPUT OPTIONS

NECPAT interactively gives the user information about the currents induced in the object and the scattered electromagnetic near and far fields. Three types of output can be created: 1) simple 1-d line plots on the EIAW screen which can also be saved in a graphics file for printing on a laser printer; 2) files for color contour plots which are viewed using the GPK; and 3) a printed output file containing the currents and all of the radiation pattern information requested for plotting. If the user wants to plot 1-d graphs directly to the Workstation screen, NECPAT is started up in the EIAW environment by choosing Create Plots from the Output Menu.

1. Far Field Information

In the far field approximation, the scattered electromagnetic field information depends on distance from the object only via a factor, $(1/R)\exp(i\mathbf{k}\cdot\mathbf{R})$, where $\mathbf{R}$ is the vector from the center of the object to a point in the far field and $\mathbf{k}$ is the wave vector. NECPAT factors out this radial dependence so that the far field information depends only on two spherical coordinates, θ and ϕ. The relationship of this spherical coordinate system to the rectangular coordinate system used in NEC, NECPAT, and PATRAN is shown in Figure 8.2.

a. Color Contour Plots. NECPAT has options for calculating and creating files for θ-ϕ color contour plots of any of the far field quantities calculated by the original NEC code. These far field quantities include total and component gains (or radar cross sections for scattering problems) and the magnitude and phase of both the θ and ϕ components of the electric fields (with the radial dependence factored out). NECPAT

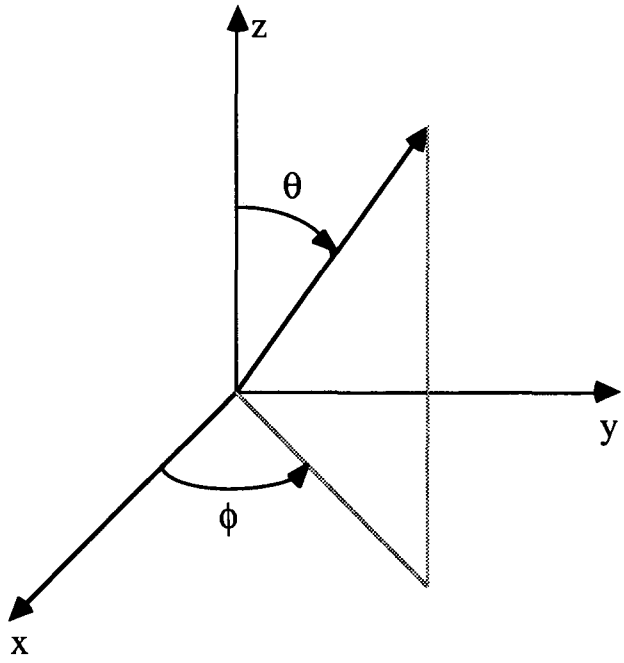


Figure 8.2. Definition of the spherical coordinate system used in NEC, NECPAT, and PATRAN

queries the user as to which region in θ and ϕ is to be plotted and the number of θ and ϕ increments to be used. NECPAT then computes all the far field information for the specified range of points in θ and ϕ and writes the information to a file.

Using the GPK, the color contour plots are actually drawn on the surface of a fully three-dimensional sphere (or portion thereof) of unit radius. Thus, if the user specifies the entire sphere (θ = 0 - 180 degrees, ϕ = 0 - 360 degrees), the GPK contour plots far field quantities directly onto the surface of the unit sphere. Using GPK commands, the user can interactively rotate this sphere to view any region in θ and ϕ. A color bar on the right of the Workstation screen gives the numerical values for the various contour levels.

In order to make these contour plots, NECPAT creates three distinct files for GPK. One file, a application-independent neutral file (see Section VIII.B.2 and Figure 7.8), contains the information needed by GPK for creating a 3-d partial or full unit sphere with nodes at the specified points in θ and ϕ. Another file, a node results file, contains the various far field quantities for the requested points (nodes) in θ and ϕ. A node results file,

like a neutral file, is a strictly formatted text file with one record (line) per node containing the node ID in the first column and up to 200 columns of results for this node.

The far field results file created by NECPAT contains nine columns of information for each node (θ-ϕ) point. These columns contain the same information (in the same order) which appears in the original NEC printed output of the radiation pattern, but without the three polarization columns. Thus, the first two columns are θ and ϕ for this node; the next three are the three gains or scattering cross sections (vertical, horizontal and total, or major, minor and total, as requested); and the last four are the magnitude and phase of the θ and ϕ components of E, respectively (see [8-1]). Upon request, GPK will create a color contour plot of any of these columns.

A third PATRAN file is also created by NECPAT: a PATRAN session file. A session file is a text file containing commands which a user would normally give to a terminal. The graphics package creates such a session file automatically each time it runs. All commands entered by the user via the terminal are saved in this file. PATRAN offers the user the option of input via session files so that long, tedious user sessions, or portions thereof, can be automatically repeated by the user without reentering the commands from the terminal input. These files are discussed in detail in the graphics software manual [8-2]. The session files created by NECPAT contain the PATRAN commands for creating the color contour plot of the total gain (or radar cross section for scattering problems) for the specified portion of the unit sphere. Once the graphic software has finished executing the commands in the session file, the user can then interactively display contour plots of any far field quantities contained in the results file. Moreover, by using PATRAN's split screen capability, comparisons of different far field quantities, or the same quantity from different solutions, can also be made.

b. 1-d Line Plots. NECPAT can create directly on the EIAW screen simple 1-d line plots of the same far field information discussed in the contour plot section, e.g., the three gains (or scattering cross sections for scattering problems) and the magnitudes and phases of both the θ and ϕ components of the far electric field. A sample 1-d plot of the magnitude of the θ component of the far electric field for ϕ = 180 - 360 degrees at θ = 90 degrees is shown in Figure 8.3. The case plotted is the test case presented in Section V. Figure 8.3 corresponds to a slice through the contour plot shown in Figure 5.14 at θ = 90 degrees. For NECPAT to create a 1-d line plot, the user must

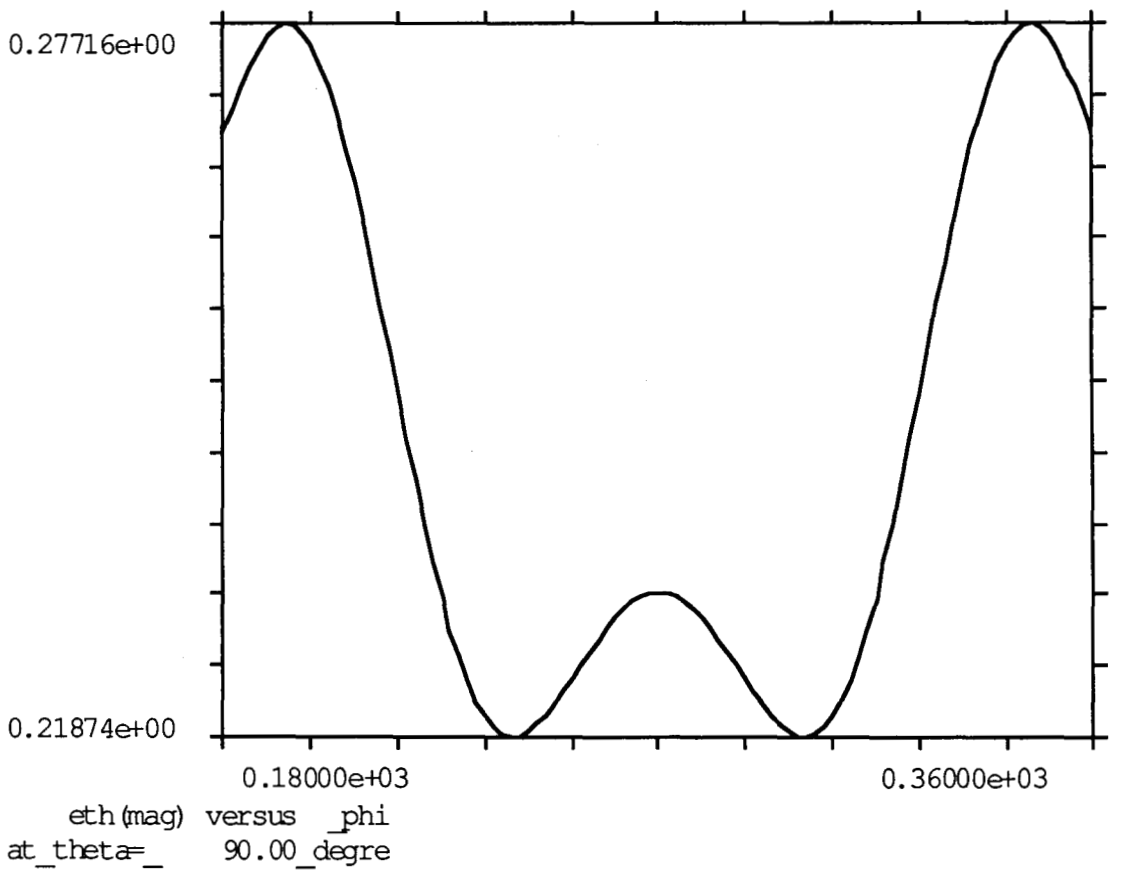


Figure 8.3. Sample 1-d far field plot for the test case of Section V

specify 1) the coordinate to vary, 2) the range of variation, 3) the value of the increment in the coordinate to vary and 4) the fixed value of the other coordinate. Any of the far field quantities can then be plotted with this choice of x-axis. If the user requested that the 1-d line plots be saved in a file, these plots can also be sent to a laser printer (or viewed again) after NECPAT is terminated. This involves the use of several other programs to translate the graphics file.

c. Printed Output. If the user has requested that a printed output file be created by NECPAT, all the far field information for the values of θ and ϕ requested for either 1-d or contour plots will also be written to the output file.

2. Near Field Information

To describe the **far** field, only two components of the electric field and two coordinates ($E\theta$, $E\phi$, θ, ϕ) are needed because the radial dependence could be factored out; all of the far field information could be contained in two unit-sphere contour plots, one of $E\theta$ for all θ and ϕ and one of $E\phi$ for all θ and ϕ. A complete description of the **near** field pattern requires knowledge of all three components of both the electric and magnetic field at every point (Ex, Ey, Ez, Hx, Hy, Hz, x, y, x). A description of the near field on just one plane (e.g., the x-y plane at a fixed z) requires six contour plots for the electric and magnetic fields. Because there is so much more information needed to describe the near field, the near field portion of NECPAT is somewhat more complicated than the far field. Moreover, the original NEC code allows the user to obtain near field information in either spherical or rectangular coordinates. NECPAT is limited to rectangular coordinates for near field output.

a. Color Contour Plots. NECPAT has options for creating files for rectangular contour plots of any component (magnitude and phase) of either the electric or magnetic field for any regions of any x-y, y-z, or z-x plane.

To generate near field contour plots, NECPAT first queries the user as to which spatial region is to be plotted and with what resolution, i.e., which plane (which fixed coordinate) and what range of the other two coordinates and with what increments. The user next must specify whether the electric or magnetic field is to be plotted (the component or phase to be plotted is not specified at this time). NECPAT then calculates all three components (magnitude and phase) of this field for the specified range of points (nodes) and puts this information in a compatible node results file. The near field results files contain six columns of results for each node, e.g., EX(magnitude), EX(phase), EY(magnitude), EY(phase), EZ(magnitude), EZ(phase), respectively, for the electric field file. NECPAT also creates a PATRAN session file for creating the grid points and the plots. NECPAT then asks the user as to whether the other field (E or H) should be calculated for this same range of points.

To view a near field contour plot, the user runs the graphics software using the NECPAT-generated session file; this file contains commands to create the grid and read in the file containing the calculated near fields. GPK can then contour plot any component or

phase specified by the user. (The near field results files contain six columns of results, e.g., EX(magnitude), EX(phase), EY(magnitude), EY(phase), EZ(magnitude), EZ(phase) for the electric field file.) Using PATRAN's split screen capability, different components or phases can be plotted in different windows. Because of PATRAN's flexibility, the user can also use the split screen to compare electric and magnetic fields or electric fields from two different runs. Moreover, the graphics software also allows vector results such as these near electric field results to be displayed as colored vectors in its three-dimensional space; the direction of the plotted vector is the direction of the near electric field and the color gives the magnitude.

b. 1-d Line Plots. NECPAT can create directly on the Workstation screen simple 1-d plots of the magnitude and phase of any rectangular component of the near electric and magnetic fields for any range of x, y or z (at any fixed values of the other two coordinates). For example, Figure 8.4 shows a line plot of the magnitude of the z component of the electric field as a function of x at y = -0.6163 and z = -0.3 for the test case in Section V. This corresponds to a slice through the contour plot of Figure 5.2 at z = -0.3. As for the far field, these plots can also be saved in a file and printed or viewed later.

The user must specify to NECPAT the coordinate to vary, the range of the coordinate to be plotted, the increment, and the fixed value of the other two coordinates. The user also specifies which field (E or H) is to be plotted. NECPAT then calculates the magnitude and phase of all three components of the specified field; the user can then request a plot of any of these. The user can then also ask NECPAT to calculate and plot the other field for this same range of points.

c. Printed Output. If the user has requested that a printed output file be created by NECPAT, the near field information for the fields (E and/or H) for the values of x, y, and z requested for either 1-d or contour plots will also be written to the output file.

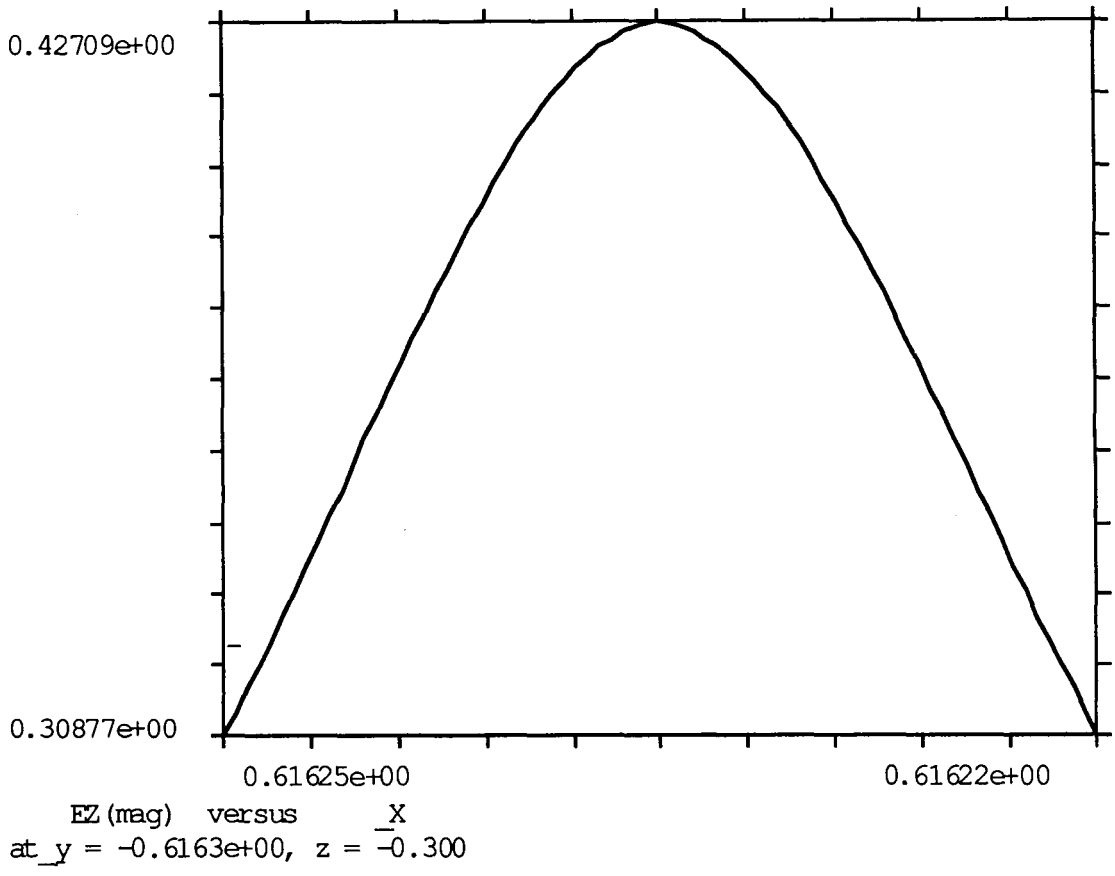


Figure 8.4. Sample 1-d near field plot for the test case of Section V

3. Induced Currents

a. Color Contour Plots. For objects modeled with quadrilateral patches only, NECPAT has an option for creating a file for a color contour plot of the magnitude of the induced currents. The GPK is used to make color contour plots of the currents displayed on the three-dimensional object itself. Within GPK, the object can be rotated, or viewed from several angles using a split screen, to see the induced currents on various portions of the object's surface.

Specifically, upon request, NECPAT creates a PATRAN-formatted element results file containing the magnitudes of the induced current in each patch of the object. The magnitude of the currents are presently in column 1. The user must also have an application-independent neutral file for the object's geometry (see Section VII.B) which

contains finite element information for each patch. To create the contour plot of the induced currents, the user runs the graphics software and 1) uses the interface mode option to read in the object's neutral file; 2) uses the results mode option to read in the element results (the magnitude of the currents); and 3) uses the contour plotting option to view the results. If the object's neutral file does not exist, it can be created from the geometry information contained in the solution file using the translator (Section VII.A). At present, this option is implemented for objects modeled with quadrilateral patches only.

b. Printed Output. Upon request, the currents in both wires and patches are put in the printed output file in the same format as in the original sequential NEC.

D. FUTURE PLANS

Several additional options for the analysis of NEC results are planned for the next year. Three additional options are planned for NECPAT: 1) Options for creating near field 1-d and 2-d plots in spherical coordinates will be added. 2) The capability of making contour plots of the induced currents on the surface of objects modeled with arbitrary and triangular patches will be added. At present this option is implemented only for objects modeled with rectangular or quadrilateral patches. 3) An option to plot the monostatic radar cross section as a function of wave number will be added. Since each wave number corresponds to a separate NEC solution, this will involve combining information from several NEC solution files.

Soon it will be possible to use the Sun workstation computer as the control processor for the hypercube. In this situation, the hypercube NEC code and the workstation NECPAT code will be merged so that the code structure will be as in the earlier hypercube NEC code (with the Sun running the CP portion). The output portion of the hypercube NEC code would then be replaced with the interactive output code from NECPAT and the requested radiation pattern information would be calculated in parallel in the elements. The user would have options for either computing the solution or reading it from a file.

REFERENCES

[8-1] G. J. Burke and A. J. Poggio, Numerical Electromagnetics Code (NEC) - Method of Moments, Lawrence Livermore National Laboratory, Livermore, CA, 1981.

[8-2] PATRAN Plus User Manual, Vols. 1 and 2, PDA Engineering, Costa Mesa, CA, July 1987.

SECTION IX

CONCLUSIONS

The first objective of the Hypercube Matrix Computation task is to investigate the applicability of a parallel computing architecture to the implementation and solution of large-scale electromagnetic scattering problems. Three analysis codes are being assessed on a parallel computing Mark III Hypercube. The first code which has been implemented is the frequency domain method of moments solution, Numerical Electromagnetics Code (NEC-2), developed at Lawrence Livermore National Laboratory. The second code is a time domain finite difference solution to Maxwell's equations. A third code currently being developed utilizes a finite element technique. The second objective of this effort is to measure the performance of the parallel electromagnetics scattering analysis codes. The third objective is to integrate the developed analysis capabilities into an Electromagnetic Interactive Analysis Workstation. The workstation has been designed to facilitate all three analysis functions: 1) graphical specification of the problem (i.e., the object to be analyzed), 2) execution of the analysis codes, and 3) graphical display of the output.

Several quantitative measures of performance may be applied when assessing the applicability of parallel architecture to large-scale electromagnetics scattering problems. First, the problem size possible on the hypercube with 128 megabytes of dynamic memory for a 32-node configuration is compared with that possible on a conventional mainframe or minicomputer. Second, the performance of the codes can be analyzed for the computational speedup attained by the parallel architecture. The speedup can be measured in at least three different ways: 1) comparing the times for the code running in 32 nodes with the times for the code running in a single node; 2) comparing the times (and therefore scalability to larger hypercube configurations) when the problem size per node is fixed and the number of nodes in use is varied; and 3) comparing the CPU times for key components of the code running on the 32-node Mark III Hypercube with the CPU times for the same code components running, for instance, on a VAX 11/750.

A. PROBLEM SIZE

In addition to providing multiple processors for increased computing speed, parallel computing architectures offer the possibility of an increase in available memory if each

node has its own memory. The VAX 11/750 has about 5 megabytes of memory to a typical user whereas the Mark III has four megabytes per node, which gives a total of 128 megabytes of dynamic RAM available on a 32-node hypercube.

For the NEC code, the size of the largest problem that can be run is determined by the largest interaction matrix which can be stored in memory. The matrix is N_T x N_T with $N_T = N + 2 \times M$ where N is the number of wire segments and M is the number of patches used to model the object. Unless the symmetry option can be invoked, the largest problem that can run on the VAX in the typical user space has 300 equations. This means that the largest object which can be modeled contains 300 wire segments or 150 patches. On the Mark III 32-node, the largest problem currently can contain as many as 2400 wires and or 1200 patches (more, if the symmetry is invoked). The total matrix which is distributed across the nodes of the hypercube then contains 2400 x 2400 double precision complex elements. This reflects a factor of 8 increase in terms of the number of patches in the modeled object relative to that on a VAX. With the availability of four disk drives, the maximum problem once the out-of-core capability is implemented is 5000 patches or 10,000 wires.

For finite difference code, the largest size problem is determined by the number of unit cells used to model the computational lattice. For the VAX 11/750, we can allocate memory for about 192,000 unit cells. For the Mark III Hypercube with 32 nodes active, we can allocate memory for about 2,048,000 unit cells.

B. HYPERCUBE PERFORMANCE

1. Code Speed on 32 Nodes Relative to 1 Node

To measure the suitability of the codes to the parallel architecture of the hypercube, we use the speedup factor, defined as the ratio of the run time on N nodes to the run time on just 1 node. If there are no penalties associated with the parallel computing decomposition, the speedup factor would be N. However, when a code is run on multiple nodes of a parallel computer, the speedup does not increase linearly with the number of nodes because there is some "overhead" associated with running in parallel. The parallel overhead can be due to several sources: time spent in communication between nodes, time

lost if the work load among the nodes is not exactly the same, and additional code for the administration of the parallel code.

The FDTD code is an excellent candidate for a parallel architecture because the finite difference method is intrinsically a "local" calculation. The computation at each grid point utilizes information from at most two unit cells. This local calculation feature permits low parallel communication overhead. Also, load imbalance is not a serious problem because we can evenly divide the global grid among the nodes. The suitability of this code for parallel computing is reflected in the high speedup factors. For the largest problem that could be run on 1 node, the speedup factor in going from 1 to 32 nodes was 25.4 (or 79.3% efficient). It is important to note that because the problem size remains the same, the number of cells in each node of a 32-node configuration is reduced by a factor of 32. The larger hypercube configurations are therefore not run at full capacity. In other words, the ratio of computation to communication times is reduced. For larger problems, where we run the same-size problem on 32 nodes that saturates an 8-node configuration, the number of cells in each processor is reduced by a factor of 4. The speedup factor in going from 8 nodes to 32 nodes is 3.7 (or 92.7% efficient).

For the NEC code, speedups are given for the two computationally intensive parts of the code separately, the matrix fill and, the Gaussian elimination factorization time. The speedup for these times summed are also given, in Section IV. The fill of the interaction matrix is also ideally suited to parallel architecture because the various rows of the matrix can be filled independently of each other, leading to very low communication overhead. However, when the number of rows is not much larger than the number of nodes, load imbalance becomes apparent in the statistics from the various nodes. Nevertheless, the speedups found for the fill are excellent. For one of the largest problems that can fit into one node (290 wire segments), the speedup factor for 32 nodes relative to 1 node is 25.1.

Because by nature matrix operations generally need information from many or all of the other matrix elements, matrix computations typically demonstrate less speedup than other algorithms, where the data internode communication requirements are low or confined to neighboring nodes. For this reason concurrent implementation of matrix algorithms requires great care. Data must be distributed so that the amount of communication is minimized. For parallel NEC, the internode data dependency is reflected in the relatively lower speedups for the factorization part compared to the fill. As shown in

Figure 4.26, an appreciable speedup factor of 21.4 has been obtained particularly for larger problems (again, 290 wire segments).

2. Fixed Problem Size

Measuring how a problem scales by comparing the execution time on a 32-node hypercube with the time on a single node tends to show a gradual drop in performance as the number of nodes in use is increased. This occurs because the ratio of computation to communication is dropping as the number of nodes is increased. Although the overall problem size has remained the same, each node has less computational work to do whereas the amount of communication has either remained the same or, in some cases, increased. For this reason, to better understand how a particular algorithm scales to larger hypercube configurations, we need to keep the computational load fixed.

The fixed problem size performance test has been performed for the FTDT code and the fill of the interaction matrix in the parallel NEC code. In both cases, once the computational load per node was held the same, the performance increased nearly linearly with the number of nodes in use. In other words, the overhead incurred in both cases for communication was small. We did not do the fixed case problem, however, for the matrix factorization in the parallel NEC code. To keep the computational load fixed per node, larger and larger matrices would need to be generated, requiring increasingly more transformation steps. In this case the communication would increase with the number of transformation steps required.

3. Comparison With Other Sequential Computers

Timing comparisons between the VAX and the Mark III 32-node for the NEC code were done for two different types of problems. One was a problem in which the object was modeled by patches (Table 4.1) and the other one a problem in which the object was modeled by wires (Table 4.2). Comparisons are provided for both because, for a fixed number of segments or patches, the matrix fill portion of the code takes more computations for wire structures than it does for surface patches. For the largest problems that can be run on the VAX, the Mark III 32-node overall is 29.6 times faster for the patch case (224 patches) and 26.4 times faster for the wire case (290 wire segments). Thus a

wire case that runs for an hour on the VAX is reduced to 2.0 to 2.3 minutes on the Mark III Hypercube. This speedup makes it feasible for the user to run the NEC code in an interactive mode, rather than in a batch mode.

Direct timing comparisons between a Mark III Hypercube using 32 active nodes and a VAX are not available for the finite difference time domain code. We can, however, make an estimate of the relative performance by executing the Taflove sequential code on a VAX and comparing it with the performance of the parallel FDTD code running on the hypercube. Both codes are similar in capabilities. For a conducting cube embedded in a 40 x 40 x 40 lattice, the Mark II Hypercube ran approximately 22.7 times faster than the Taflove code running on a VAX 11/750 and 8.8 times faster than that code running on a VAX 11/785.

C. ELECTROMAGNETIC INTERACTIVE ANALYSIS WORKSTATION

The hypercube has reduced the execution times for the analysis codes so significantly that even moderate-size problems can now be run interactively. To ease the specification of the scattering or radiating structure and to simplify the interpretation of the output, we have embedded the analysis codes in a workstation environment which combines the computational power of the Mark III Hypercube with the color graphics and user-friendly multi-window environment available on a Sun Color Graphics workstation. The Electromagnetic Interactive Analysis Workstation allows the user to specify an object to be analyzed by graphical object specification or by interacting with an application-intelligent editor. Hypercube execution is invoked directly from within the workstation environment. The output may be analyzed by requesting any of a host of 2-d and 3-d graphical representations of such information as the currents, near fields, far fields, or radar cross sections.

D. CURRENT WORK

In the finite element part of our electromagnetics analysis development, the objective is to evaluate candidate matrix algorithms for parallel application in finite element scattering problems on the hypercube. A family of iterative solution methods based on the method of conjugate gradients has been demonstrated and shown to be amenable to

efficient parallel computation. Preliminary work on the development of the parallel program is under way, and efforts in the near future will include the concurrent solution of larger problems of practical interest, as well as enhanced matrix solution and domain decomposition algorithms.

The parallel method of moments code (NEC) is being enhanced by the addition of an iterative implementation of a Numerical Green's Function which will allow incremental design of a structure. The addition of disk drives attached directly to hypercube nodes allows the storage of previously factored interaction matrices which can be reloaded for subsequent solutions. In addition, the disk drives allow larger problems to be solved by permitting out-of-core solutions.

Although all of the timing runs presented in this report were made using the Motorola 68020 math co-processor, the Motorola 68881, two 32-node hypercubes have now become available with the floating point accelerator daughterboards added. In the fall these 32-bit floating point units are scheduled to be upgraded to support 64-bit arithmetic. In addition, by January 1989 the 128-node Mark III Hypercube will be available. Soon we plan to update the performance results for the test cases that we have reported here by using the newly accelerated hypercubes.

We have demonstrated that the hypercube is very well suited to the solution of large-scale electromagnetic scattering problems. Three different analysis codes are being assessed: time domain finite difference, frequency domain method of moments, and frequency domain finite elements. Each of the codes makes use of different numerical algorithms and thereby provides a demonstration of the flexibility of the hypercube architecture to different applications. The 32-node Mark III Hypercube has been used to concurrently solve electromagnetic scattering problems too large and/or too time consuming to be done on a sequential computer such as a VAX. Our ability to measure and characterize the performance for variable-size hypercube configurations plus the additions of the very high speed floating point processor daughterboards and of the concurrent input/output devices allow us to extrapolate to the performance of even larger configurations on larger problems.

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. JPL Pub. 88-31	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Hypercube Matrix Computation Task		5. Report Date August 1, 1988 6. Performing Organization Code
7. Author(s) Ruel H. Calalo, et al.		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91109		10. Work Unit No. 11. Contract or Grant No. NAS7-918 13. Type of Report and Period Covered JPL Publication
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code RE232 PX-644-11-00-00-08

15. Supplementary Notes

16. Abstract

A major objective of the Hypercube Matrix Computation effort at the Jet Propulsion Laboratory (JPL) is to investigate the applicability of a parallel computing architecture to the solution of large-scale electromagnetic scattering problems. Three scattering analysis codes are being implemented and assessed on a JPL/California Institute of Technology (Caltech) Mark III Hypercube. The codes, which utilize different underlying algortithms, give a means of evaluating the general applicability of this parallel architecture. The three analysis codes being implemented are a frequency domain method of moments code, a time domain finite difference code, and a frequency domain finite elements code. These analysis capabilities are being integrated into an electromagnetics interactive analysis workstation which can serve as a design tool for the construction of antennas and other radiating or scattering structures.

This document is a summary of the first two years of work on the Hypercube Matrix Computation effort. It includes both new developments and results as well as work previously reported in the "Hypercube Matrix Computation Task: Final Report for 1986-87" (JPL Publication 87-18).

17. Key Words (Selected by Author(s))	18. Distribution Statement
Computers Computing Communications	Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	183	